IBM Tivoli Storage Manager
Version 7.1.6

Using the Application Programming Interface

IBM Tivoli Storage Manager
Version 7.1.6

Using the Application Programming Interface

Note:
Before you use this information and the product it supports, read the information in "Notices" on page 209.

Fifth edition (June 2016)

This edition applies to version 7, release 1, modification 6 of IBM Tivoli Storage Manager (product numbers 5608-E01, 5608-E02, and 5608-E03) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1993, 2016.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this publication v
Who should read this publication v
Publications v
Conventions used in this publication v

What's new in Version 7.1.6. vii

Chapter 1. API overview 1
Understanding configuration and options files . . . 1
Setting up the API environment 3

Chapter 2. Building and running the sample API application 5
UNIX or Linux sample application source files . . . 5
Building the UNIX or Linux sample application . 6
Windows 32-bit sample application 7
Windows 64-bit sample application 8

Chapter 3. Considerations for designing an application 11
Determining size limits 14
Maintaining API version control 14
Using multithreading 16
Signals and signal handlers 16
Starting or ending a session 17
Session security 18
Setting the `passwordaccess` option to `generate` without TCA 21
Creating an administrative user with client owner authority 22
Object names and IDs 23
File space name 23
High-level and low-level names 24
Object type 24
Accessing objects as session owner 25
Accessing objects across nodes and owners 25
Managing file spaces 26
Associating objects with management classes . . . 28
Query management classes 30
Expiration/deletion hold and release 30
Archive data retention protection 31
Querying the Tivoli Storage Manager system . . . 33
Example of querying the system 34
Server efficiency 35
Sending data to a server 36
The transaction model 36
File aggregation 37
LAN-free data transfer 37
Simultaneous-write operations 37
Enhancing API performance 38
Set up the API to send performance data to the client performance monitor 38
Configuring client performance monitor options 39
Sending objects to the server 41
Understanding backup and archive objects . . . 41
Compression 42
Buffer copy elimination 44
API encryption 46
Data deduplication 50
API client-side data deduplication 52
Server-side data deduplication 55
Application failover 55
Failover status information 56
Example flow diagrams for backup and archive . . 58
Code example of API functions that send data to Tivoli Storage Manager storage 61
File grouping 63
Receiving data from a server 65
Partial object restore or retrieve 66
Restoring or retrieving data 66
Example flow diagrams for restore and retrieve 70
Code example of receiving data from a server . . 72
Updating and deleting objects on the server . . . 73
Deleting objects from the server 73
Logging events 74
State diagram summary for the Tivoli Storage Manager API 74

Chapter 4. Understanding interoperability 77
Backup-archive client interoperability 77
Naming your API objects 77
Backup-archive client commands you can use with the API 79
Operating system interoperability 80
Backing up multiple nodes with client node proxy support 80

Chapter 5. Using the API with Unicode 83
When to use Unicode 83
Setting up Unicode 83

Chapter 6. API function calls 85
dsmBeginGetData 87
dsmBeginQuery 89
dsmBeginTxn 93
dsmBindMC 94
dsmChangePW 95
dsmCleanUp 96
dsmDeleteAccess 97
dsmDeleteFS 97
dsmDeleteObj 98
dsmEndGetData 99
dsmEndGetDataEx 100
dsmEndGetObj 100
dsmEndQuery 101
dsmEndSendObj 101
dsmEndSendObjEx 102
dsmEndTxn 102

© Copyright IBM Corp. 1993, 2016

dsmEndTxnEx 104
dsmGetData 105
dsmGetBufferData 106
dsmGetNextQObj 107
dsmGetObj 110
dsmGroupHandler 111
dsmInit 112
dsmInitEx 115
dsmLogEvent 119
dsmLogEventEx 120
dsmQueryAccess 121
dsmQueryApiVersion 122
dsmQueryApiVersionEx 122
dsmQueryCliOptions 123
dsmQuerySessInfo 124
dsmQuerySessOptions 125
dsmRCMsg 126
dsmRegisterFS 127
dsmReleaseBuffer 128
dsmRenameObj 128
dsmRequestBuffer 130
dsmRetentionEvent 131
dsmSendBufferData 132
dsmSendData 133
dsmSendObj 134
dsmSetAccess 137
dsmSetUp 138
dsmTerminate 140
dsmUpdateFS 140
dsmUpdateObj 141
dsmUpdateObjEx 142

Appendix A. API return codes source file: `dsmrc.h` 145

Appendix B. API type definitions source files 155

Appendix C. API function definitions source file 197

Appendix D. Accessibility features for the Tivoli Storage Manager product family 207

Notices 209

Glossary 213

Index 215

About this publication

This publication provides information to help you to perform the following tasks:
- Add IBM® Tivoli® Storage Manager application program interface calls to an existing application
- Write programs with general-use program interfaces that obtain the services of Tivoli Storage Manager.

In addition to the application programming interface (API), the following programs are included on several operating systems:
- A backup-archive client program that backs up and archives files from your workstation or file server to storage, and restores and retrieves backup versions and archived copies of files to your local file systems.
- A Web backup-archive client that an authorized administrator, support person, or end user can use to perform backup, restore, archive, and retrieve services using a Web browser on a remote machine.
- An administrative client program that you can access from a Web browser or from the command line. An administrator controls and monitors server activities, defines storage management policies for backup, archive, and space management services, and sets up schedules to perform these services at regular intervals.

Who should read this publication

This publication provides instructions for you to add API calls to an existing application. You should be familiar with C programming language and Tivoli Storage Manager functions.

Publications

The Tivoli Storage Manager product family includes IBM Tivoli Storage FlashCopy® Manager, IBM Tivoli Storage Manager for Space Management, IBM Tivoli Storage Manager for Databases, and several other storage management products from IBM.

To view IBM product documentation, see IBM Knowledge Center.

Conventions used in this publication

This publication uses the following typographical conventions:

Example	Description
autoexec.ncf hsmgui.exe	A series of lowercase letters with an extension indicates program file names.
DSMI_DIR	A series of uppercase letters indicates return codes and other values.
dsmQuerySessInfo	Boldface type indicates a command that you type on a command line, the name of a function call, the name of a structure, a field within a structure, or a parameter.
timeformat	Boldface italic type indicates a Tivoli Storage Manager option. The bold type is used to introduce the option, or used in an example.

© Copyright IBM Corp. 1993, 2016

Example	Description
dateformat	Italic type indicates an option, the value of an option, a new term, a placeholder for information you provide, or for special emphasis in the text.
`maxcmdretries`	Monospace type indicates fragments of a program or information as it might appear on a display screen, such a command example.
plus sign (+)	A plus sign between two keys indicates that you press both keys at the same time.

What's new in Version 7.1.6

IBM Tivoli Storage Manager Version 7.1.6 introduces new features and updates.

For a list of new features and updates in this release and previous Version 7 releases, see API updates.

New and changed information in this product documentation is indicated by a vertical bar (|) to the left of the change.

© Copyright IBM Corp. 1993, 2016

Chapter 1. API overview

The IBM Tivoli Storage Manager application program interface (API) enables an application client to use storage management functions.

The API includes function calls that you can use in an application to perform the following operations:
- Start or end a session
- Assign management classes to objects before they are stored on a server
- Back up or archive objects to a server
- Restore or retrieve objects from a server
- Query the server for information about stored objects
- Manage file spaces
- Send retention events

When you, as an application developer, install the API, you receive the files that an end user of an application needs:
- The API shared library.
- The messages file.
- The sample client options files.
- The source code for the API header files that your application needs.
- The source code for a sample application, and the makefile to build it.
- The dsmtca file (UNIX and Linux only) .

For 64-bit applications, all compiles should be performed using compiler options that enable 64-bit support. For example, '-q64' should be used when building API applications on AIX®, and '-m64' should be used on Linux. See the sample make files for more information.

Important: When you install the API, ensure that all files are at the same level.

For information about installing the API, see Installing the Tivoli Storage Manager backup-archive clients.

References to UNIX and Linux include AIX, HP-UX, Linux, Mac OS X, and Oracle Solaris operating systems.

Understanding configuration and options files

Configuration and options files set the conditions and boundaries under which your session runs.

You, an administrator, or an end user can set option values to:
- Set up the connection to a server
- Control which objects are sent to the server and the management class to which they are associated

You define options in one or two files when you install the API on your workstation.

On UNIX and Linux operating systems, the options reside in two options files:
- dsm.opt - the client options file
- dsm.sys - the client system options file

© Copyright IBM Corp. 1993, 2016

On other operating systems, the client options file (dsm.opt) contains all of the options.

Restriction: The API does not support the following backup-archive client options:
- `autofsrename`
- `changingretries`
- `domain`
- `eventlogging`
- `groups`
- `subdir`
- `users`
- `virtualmountpoint`

You also can specify options on the **dsmInitEx** function call. Use the option string parameter or the API configuration file parameter.

The same option can derive from more than one configuration source. When this happens, the source with the highest priority takes precedence. Table 1 lists the priority sequence.

Table 1. Configuration sources in order of decreasing priority

Priority	UNIX and Linux	Windows	Description
1	dsm.sys file (client system options)	not applicable	This file contains options that a system administrator sets only for UNIX and Linux. **Tip:** If your dsm.sys file contains server stanzas, ensure that the `passwordaccess` option specifies the same value (either `prompt` or `generate`) in each of the stanzas.
2	Option string (client options)	Option string (all options)	One of these options takes effect when it is passed as a parameter to a **dsmInitEx** call. The list can contain client options such as `compressalways`, `servername` (UNIX and Linux only), or `tcpserveraddr` (non-UNIX). With the API option string, an application client can make changes to the option values in the API configuration file and the client options file. For example, your application might query the end user if compression is required. Depending on the user responses, you can construct an API option string with this option and pass it into the call to **dsmInitEx**. For information about the API option string format, see "**dsmInitEx**" on page 115. You also can set this parameter to `NULL`. This indicates that there is no API option string for this session.
3	API configuration file (client options)	API configuration file (all options)	The values that you set in the API configuration file override the values that you set in the client options file. Set up the options in the API configuration file with values that are appropriate in the Tivoli Storage Manager session for the user. The values take effect when the API configuration file name is passed as a parameter in the **dsmInitEx** call. You also can set this parameter to NULL. This indicates that there is no API configuration file for this session.

Table 1. Configuration sources in order of decreasing priority (continued)

Priority	UNIX and Linux	Windows	Description
4	dsm.opt file (client options)	dsm.opt file (all options)	On UNIX and Linux operating systems the dsm.opt file contains the user options only. On other operating systems, the dsm.opt file contains all options. To override the options in these files, follow the methods that are described in this table.

Related concepts:

Processing options

Setting up the API environment

The API uses unique environment variables to locate files. You can use different files for API applications from those that the backup-archive client uses. Applications can use the **dsmSetup** function call to override the values that the environment variables set.

Tip: On Windows, the default installation directory is: `%SystemDrive%\Program Files\Common Files\Tivoli\TSM\api`

Table 2 lists the API environment variables by operating system.

Table 2. API environment variables

Variables	UNIX and Linux	Windows
DSMI_CONFIG	The fully-qualified name for the client options file (dsm.opt).	The fully-qualified name for the client options file (dsm.opt).
DSMI_DIR	Points to the path that contains the dsm.sys, dsmtca, en_US subdirectory, and any other national language support (NLS) language. The en_US subdirectory must contain dsmclientV3.cat.	Points to the path that contains dscenu.txt and any NLS message file.
DSMI_LOG	Points to the path for the dsierror.log file.	Points to the path for the dsierror.log file. If the client **errorlogname** option is set, the location specified by that option overrides the directory specified by DSMI_LOG.

Chapter 2. Building and running the sample API application

The API package includes sample applications that demonstrate the API function calls in context. Install a sample application and review the source code to understand how you can use the function calls.

Select one of the following sample API application packages:
- The interactive, single-threaded application package (dapi*)
- The multithreaded application package (callmt*)
- The logical object grouping test application (dsmgrp*)
- The event-based retention policy sample application (callevnt)
- The deletion hold sample application (callhold)
- The data retention protection sample application (callret)
- The IBM Tivoli Storage Manager buffer sample program (callbuff)

To help you get started, review the procedure to build the sample dapismp sample application by your platform:
- For UNIX or Linux applications, see "UNIX or Linux sample application source files."
- For Windows applications, see "Windows 32-bit sample application" on page 7, or "Windows 64-bit sample application" on page 8.

The dapismp sample application creates its own data streams when backing up or archiving objects. It does not read or write objects to the local disk file system. The object name does not correspond to any file on your workstation. The "seed string" that you issue generates a pattern that can be verified when the object is restored or retrieved. Once you compile the sample application and run **dapismp** to start it, follow the instructions that display on your screen.

UNIX or Linux sample application source files

To build and run the sample UNIX or Linux sample application, you need to ensure you have certain source files. Once you build the sample application you can compile and run it.

The files that are listed in Table 3 include the source files and other files that you need to build the sample application that is included with the API package.

Table 3. Files that you need to build the UNIX or Linux API sample application

File names	Description
README_api_enu	README file
dsmrc.h	Return codes header file
dsmapitd.h	Common type definitions header file
dsmapips.h	Operating system-specific type definitions header file
dsmapifp.h	Function prototype header file
release.h	Release values header file

© Copyright IBM Corp. 1993, 2016

Table 3. Files that you need to build the UNIX or Linux API sample application (continued)

File names		Description
dapibkup.c dapidata.h dapiinit.c dapint64.h dapint64.c dapipref.c dapiproc.c dapiproc.h	dapipw.c dapiqry.c dapirc.c dapismp.c dapitype.h dapiutil.h dapiutil.c	Modules for the command line-driven sample application
makesmp[64].*xxx*		Makefile to build dapismp for your operating system. The *xxx* indicates the operating system.
callmt1.c callmt2.c		Multi-threaded sample files
callmtu1.c callmtu2.c		Multi-threaded Unicode sample files
libApiDS.xx libApiDS64.xx, or libApiTSM64.xx		Shared library (the suffix is platform-dependent)
dsmgrp.c callevnt.c callhold.c callret.c callbuff.c dpsthread.c		Grouping sample files Event-based retention policy sample source code Deletion hold sample source code Data retention protection sample source code

Building the UNIX or Linux sample application

You build the **dapismp** sample API application by using a compiler for your operating system.

You must install the following compilers to build the UNIX or Linux API sample application:

- IBM AIX - IBM Visual Age compiler Version 6 or later
- HP-IA64 - aCC compiler A.05.50 or later
- Linux - GCC compiler Version 3.3.3 or later
- Mac OS X - GCC compiler Version 4.0 or later
- Oracle Solaris - Oracle Studio C++ compiler Version 11 or later

1. To build the API samples, run the following command:

   ```
   gmake -f makesmp[64].xxx
   ```

   Where *xxx* indicates the operating system.
2. After you build the samples, set up your environment variables, including the DSMI_DIR, and your options files. For more information, see "Understanding configuration and options files" on page 1.
3. The first time you log on, log on as the root user to register your password.

   Tip: Setting the compressalways option to no might not resend an object uncompressed. This behavior depends on the application functionality.

To specify the Shared Memory communications method on AIX, the IBM Tivoli Storage Manager API client user must comply with one of the following conditions:

- Must be logged in as the root user.
- Must have the same UID as the process that is running the Tivoli Storage Manager server.

This restriction does not apply if the `passwordaccess` option is set to `generate` in the client systems option file `dsm.sys` and the TCA is being used or if you alter your application program file permissions by using the following commands:

```
chown root.system your_api_program
chown u+s your_api_program
```

For more information, see the application program documentation.

4. Run the **dapismp** command to start the application.
5. Choose from the list of options that is displayed. Ensure that you run the sign-on action before you run any other actions.

   Requirement: Always prefix the file space, high-level, and low-level names with the correct path delimiter (/) when you enter the name, for example: `/myfilespace`. You must use this prefix even when you specify the asterisk (*) wildcard character.

Related concepts:

Environment variables (UNIX and Linux systems)

Windows 32-bit sample application

To build and run the sample Microsoft Windows 32–bit application, you must install the IBM Tivoli Storage Manager API and ensure that you have certain source files.

Restrictions:

- For Windows applications that are built with V3.1 of the API, replace adsmv3.dll with the new adsmv3.dll and add in tsmapi.dll. For new applications, build the application with the tsmapi.dll. These DLLs are 32-bit DLLs.
- For best results, use dynamic loading. For an example, see the file `dynaload.c` and the implementation in the sample code.
- The `api\obj` directory contains the API sample program object files.
- Use the Microsoft C/C++ Compiler Version 15 and the makefile `makesmp.mak` to compile the API sample application **dapismp**. You might have to adjust the makefiles to your environment, specifically, the library or the include directories.
- After you compile the application, run the sample application by issuing the command **dapismp** from the `api\samprun` directory. The **dapismp** sample program contains the execution directory.
- Choose from the list of options that are displayed. Ensure that you run the sign-on action before you run any other actions.
- Always prefix the file space, high-level, and low-level names with the correct path delimiter (\) when you enter the name, for example: `\myfilespace`. You must use this prefix even when you specify the asterisk (*) wildcard character.

For Windows operating systems, the source files that you must have to build the sample application are listed in Table 4. The sample application is included in the API package. For convenience, a precompiled executable `dapismp.exe` is also included.

Table 4. Files for building the Windows 32–bit API sample application

File names	Description
api.txt	README file
`tsmapi.dll` `adsmv3.dll`	API DLLs

`dsmrc.h`
`dsmapitd.h`
`dsmapips.h`
`dsmapifp.h`
`dsmapidl.h`
`release.h`
	Return codes header file
	Common type definitions header file
	Operating system-specific type definitions header file
	Function prototype header file
	Dynamically loaded function prototype header file
	Release values header file

`dapidata.h`
`dapint64.h`
`dapitype.h`
`dapiutil.h`
	Source code header files

`tsmapi.lib`	Implicit library

`dapibkup.c`
`dapiinit.c`
`dapint64.c`
`dapipref.c`
`dapiproc.c`
`dapiproc.h`
	Source code files for `dapismp.exe`

`makesmp.mak`	Makefile for building sample applications

`callmt1.c`
`callmt2.c`
`callmtu1.c`
`callmtu2.c`
	Multi-threaded sample file
	
	Multi-threaded Unicode sample files

`callevnt.c`
`callhold.c`
`callret.c`
`dpsthread.c`
	Event-Based retention policy source code
	Deletion hold sample source code
	Data retention protection sample source code
	Threading utility source code

Windows 64-bit sample application

To build and run the sample application for Microsoft Windows 64–bit systems, you must install the IBM Tivoli Storage Manager API and ensure that you have certain source files.

Restrictions:

- For best results, use dynamic loading. For an example, see the file `dynaload.c` and the implementation in the sample code.
- Files for the sample application are in the following directories:

  `api64\obj`
  Contains the API sample program object files.

  `api64\samprun`
  Contains the sample program **`dapismp`**. The sample program contains the execution directory.

- The DLL tsmapi64.dll is a 64-bit DLL.
- Use the Microsoft C/C++ Compiler Version 15 and the makefile `makesmp64.mak` to compile the API sample application **`dapismp`**. You might have to adjust the makefiles to fit your environment, specifically the library or the include directories.
- After you compile the application, run the sample application by issuing the command **`dapismp`** from the `api64\samprun` directory.
- Choose from the list of options displayed that are displayed. Ensure that you run the sign-on action before you run any other actions.
- Always prefix the file space, high-level, and low-level names with the correct path delimiter (\) when you enter the name, for example: `\myfilespace`. You must use this prefix even when you specify the asterisk (*) wildcard character.

For Windows operating systems, the source files that you must have to build the sample application are listed in Table 5. The sample application is included in the API package. For your convenience, a precompiled executable (dapismp.exe) is also included.

Table 5. Files for building the Windows 64-bit API sample application

File names	Description
`api.txt`	README file
`tsmapi64.dll`	API DLLs

`dsmrc.h`
`dsmapitd.h`
`dsmapips.h`
`dsmapifp.h`
`dsmapidl.h`
`release.h`
	Return codes header file
	Common type definitions header file
	Operating system-specific type definitions header file
	Function prototype header file
	Dynamically loaded function prototype header file
	Release values header file

`dapidata.h`
`dapint64.h`
`dapitype.h`
`dapiutil.h`
	Source code header files

`tsmapi64.lib`	Implicit library

`dapibkup.c`
`dapiinit.c`
`dapint64.c`
`dapipref.c`
`dapiproc.c`
`dapiproc.h`
`dapipw.c`
`dapiqry.c`
`dapirc.c`
`dapismp64.c`
`dapiutil.c`
`dynaload.c`
	Source code files for dapismp.exe

`makesmpx64.mak`
(Windows x64)
`makesmp64.mak`
(Windows IA64)
	Makefiles to build sample applications

Table 5. Files for building the Windows 64-bit API sample application (continued)

File names	Description

`callmt1.c`
`callmt2.c`
`callmtu164.c`
`callmtu264.c`
	Multithreaded sample files

`dpsthread.c`	Sample file source code
`callevnt.c`	Event-Based retention policy source code
`callhold.c`	Deletion hold sample source code
`callret.c`	Data retention protection sample source code
`callbuff.c`	Shared buffer (no copy) sample source code.

Chapter 3. Considerations for designing an application

When you design an application, you must have a broad understanding of many aspects of the API.

To gain an understanding of the API, review the following topics:

- "Determining size limits" on page 14
- "Maintaining API version control" on page 14
- "Using multithreading" on page 16
- "Signals and signal handlers" on page 16
- "Starting or ending a session" on page 17
- "Object names and IDs" on page 23
- "Setting the `passwordaccess` option to `generate` without TCA" on page 21
- "Accessing objects as session owner" on page 25
- "Accessing objects across nodes and owners" on page 25
- "Managing file spaces" on page 26
- "Associating objects with management classes" on page 28
- "Expiration/deletion hold and release" on page 30
- "Querying the Tivoli Storage Manager system" on page 33
- "Sending data to a server" on page 36
- "Example flow diagrams for backup and archive" on page 58
- "File grouping" on page 63
- "State diagram summary for the Tivoli Storage Manager API" on page 74

When you design your application, review the considerations in Table 6. Start structures with **memset** fields might change in subsequent releases. The `stVersion` value increments with each product enhancement.

Table 6. API Considerations for designing an application

Design item	Considerations
Setting locale	The application must set the locale before the API is called. To set the locale to the default value, add the following code to the application: `setlocale(LC_ALL,"");` To set the locale to another value, use the same call with the proper locale in the second parameter. Check for specifics in the documentation for each operating system that you are using.

© Copyright IBM Corp. 1993, 2016

Table 6. API Considerations for designing an application (continued)

Design item	Considerations

Session control	Apply the following guidelines to session control:
	• Assign a unique node name for each IBM Tivoli Storage Manager backup-archive client and Tivoli Storage Manager API client product that you use. The following products are examples of these clients:
	– Tivoli Storage Manager for Mail
	– or Tivoli Storage Manager HSM for Windows
	• Use a consistent owner name across a backup and restore procedure.
	• Use the `passwordaccess` option to manage access to the protected password file. This option affects the use of the TCA child process on UNIX and Linux only, for node name, session owner name, and password management.
	• Ensure that sessions for data movement end when the task is completed so that devices on the server are freed for use by other sessions.
	• To permit LAN-free data transfer, use the **dsmSetup** function call with the `multithread` flag set to on.
	• On AIX, when you are using multithreaded applications or LAN-free, especially running on machines with multiple processors, set the environment variable `AIXTHREAD_SCOPE` to `S` in the environment before you start the application, for better performance and more solid scheduling. For example:
	`EXPORT AIXTHREAD_SCOPE=S`
	By setting `AIXTHREAD_SCOPE` to `S`, user threads that are created with default attributes are placed into system-wide contention scope. If a user thread is created with system-wide contention scope, the user thread is bound to a kernel thread and is scheduled by the kernel. The underlying kernel thread is not shared with any other user thread. For more information about this environment variable, see the following topic:
	"Using multithreading" on page 16
	• Ensure that only one thread in a session calls any API function at any time. Applications that use multiple threads with the same session handle must synchronize the API calls. For example, use a **mutex** to synchronize API calls:
	getTSMMutex()
	issue TSM API call
	releaseTSMMutex()
	Use this approach only when the threads share a handle. You can use parallel calls to API functions if the calls have different session handles.
	• Implement a threaded consumer/producer model for data movement. API calls are synchronous and the calls for **dsmGetData function** and **dsmSendData function** block until they are finished. By using a consumer/producer model, the application can read the next buffer during waiting periods for the network. Also, decoupling the data read/write and the network increases performance when there is a network bottleneck or delays. In general, the following holds:
	`Data thread <---> shared queue of buffers <---> communication thread (issue calls to the TSM API)`
	• Use the same session for multiple operations to avoid incurring an overhead. For applications that deal with many small objects, implement session-pooling so that the same session can be used across multiple small operations. An overhead is associated with opening and closing a session to the Tivoli Storage Manager server. The `dsmInit/dsmInitEX` call is serialized so even in a multithreaded application only one thread can sign on at any time. Also, during sign-on the API sends a number of one-time queries to the server so that the server can do all operations. These queries include policy, option, file spaces, and local configuration.

Table 6. API Considerations for designing an application (continued)

Design item	Considerations
Operation sequence	The Tivoli Storage Manager server locks file space database entries during some operations. The following rules apply when you are designing Tivoli Storage Manager API applications: • Queries lock the file space during the entire transaction. • The query lock can be shared with other query operations, so multiple query operations on the same file space can execute concurrently. • The following operations are used to modify the Tivoli Storage Manager server database (**DB Chg**): send, get, rename, update, and delete. • Completion of a **DB Chg** operation requires a file space lock during the database change at the end of the transaction. • Multiple **DB Chg** operations on the same file space can execute concurrently. There might be a delay while the sequence waits for the lock at the end transaction. • The query lock cannot be shared with **DB Chg** operations. A **DB Chg** operation delays the beginning of a query on the same file space, so design your applications to separate and serialize queries from **DB Chg** operations on the same file space.
Object naming	When you name objects, consider the following factors: • The specific object names are the high-level and low-level object names. If a unique identifier, such as a date stamp, is included in the name, then backup objects are always active. The objects expire only when they are marked inactive by the **dsmDeleteObj** function call. • The restore method for objects determines how to format the name for easy queries. If you plan to use a partial object restore (POR), you cannot use compression. To suppress compression, use the **dsmSendObj objAttr objCompressed=bTrue** function.
Object grouping	Group objects logically by using file spaces. A file space is a container on the server that provides a grouping category for the objects. The API queries all file spaces during the initial sign-on and also during queries, so the number of file spaces must be restricted. A reasonable assumption is that an application sets up 20 - 100 file spaces per node. The API can cater for more file spaces, but each file space incurs an overhead for the session. To create a more granular separation, use the `directory` object in the application.
Object handling	Do not store `objectID` values to use for future restores. These values are not guaranteed to be persistent during the life of the object. During a restore, pay special attention to the restore order. After the query, sort on this value before the restore. If you are using multiple types of serial media, then access the different types of media in separate sessions. For more information, see the following topic: "Selecting and sorting objects by restore order" on page 67
Management class	Consider how much control the application must have over the management class that is associated with the application objects. You can define include statements, or you can specify a name on the **dsmSendObj** function call.
Object size	Tivoli Storage Manager needs to know a size estimate for each object. Consider how your application estimates the size of an object. An overestimation of the object size is better than an underestimation.

Determining size limits

Certain data structures or fields in the API have size limits. These structures are often names or other text fields that cannot exceed a predetermined length.

The following fields are examples of data structures that have size limits:
- Application type
- Archive description
- Copy group destination
- Copy group name
- File space information
- Management class name
- Object owner name
- Password

These limits are defined as constants within the header file `dsmapitd.h`. Any storage allocation is based on these constants rather than on numbers that you enter. For more information, see Appendix B, "API type definitions source files," on page 155.

Maintaining API version control

All APIs have some form of version control. The API version that you use in your application must be compatible with the version of the API library that is installed on the user workstation.

The **dsmQueryApiVersionEx** should be the first API call that you enter when you use the API. This call performs the following tasks:
- Confirms that the API library is installed and available on the end user's system
- Returns the version level of the API library that the application accesses

The API is designed to be upwardly compatible. Applications that are written to older versions or releases of the API library operate correctly when you run a later version.

Determining the release of the API library is very important because some releases might have different memory requirements and data structure definitions. Downward compatibility is unlikely. See Table 7 for information about your platform.

Table 7. Platform compatibility information

Platform	Description
Windows	The message files must be at the same level as the library (DLL). The Trusted Communication Agent module (dsmtca) is not used.
UNIX or Linux	The API library, the Trusted Communication Agent module (dsmtca), and the message files must be at the same level.

The **dsmQueryApiVersionEx** call returns the version of the API library that is installed on the end user workstation. You can then compare the returned value with the version of the API that the application client is using.

The API version number of the application client is entered in the compiled object code as a set of four constants defined in dsmapitd.h:

```
DSM_API_VERSION
DSM_API_RELEASE
DSM_API_LEVEL
DSM_API_SUB_LEVEL
```

See Appendix B, “API type definitions source files,” on page 155.

The API version of the application client should be less than, or equal to, the API library that is installed on the user's system. Be careful about any other condition. You can enter the **dsmQueryApiVersionEx** call at any time, whether the API session has been started or not.

Data structures that the API uses also have version control information in them. Structures have version information as the first field. As enhancements are made to structures, the version number is increased. When initializing the version field, use the defined structure Version value in dsmapitd.h.

Figure 1 demonstrates the type definition of the structure, **dsmApiVersionEx** from the header file, dsmapitd.h. The example then defines a global variable that is named ***apiLibVer***. It also demonstrates how you can use it in a call to **dsmQueryApiVersionEx** to return the version of the end user's API library. Finally, the returned value is compared to the API version number of the application client.

```
typedef struct
{
        dsUint16_t stVersion;     /* Structure version               */
        dsUint16_t version;       /* API version                     */
        dsUint16_t release;       /* API release                     */
        dsUint16_t level;         /* API level                       */
        dsUint16_t subLevel;      /* API sub level                   */
} dsmApiVersionEx;

dsmApiVersionEx apiLibVer;

memset(&apiLibVer,0x00,sizeof(dsmApiVersionEx));
dsmQueryApiVersionEx(&apiLibVer);

/* check for compatibility problems */
dsInt16_t appVersion= 0, libVersion = 0;
  appVersion=(DSM_API_VERSION * 10000)+(DSM_API_RELEASE * 1000) +
              (DSM_API_LEVEL * 100) + (DSM_API_SUBLEVEL);
  libVersion = (apiLibVer.version * 10000) + (apiLibVer.release * 1000) +
                  (apiLibVer.level * 100) + (apiLibVer.subLevel);
   if (libVersion < appVersion)
   {
      printf("\n*************************************************************\n");
      printf("The TSM API library is lower than the application version\n");
      printf("Install the current library version.\n");
      printf("*************************************************************\n");
      return 0;
   }

printf("* API Library Version = %d.%d.%d.%d  *\n",
     apiLibVer.version,
     apiLibVer.release,
     apiLibVer.level,
     apiLibVer.subLevel);
```

Figure 1. An example of obtaining the version level of the API

Using multithreading

The multithreaded API permits applications to create multiple sessions with the IBM Tivoli Storage Manager server within the same process. The API can be entered again. Any calls can run in parallel from within different threads.

Tip: When you run applications that assume a multithreaded API, use the **dsmQueryAPIVersionEx** call.

To run the API in multithreaded mode, set the *mtflag* value to DSM_MULTITHREAD on the **dsmSetUp** call. The **dsmSetUp** call must be the first call after the **dsmQueryAPIVersionEx** call. This call must return before any thread calls the **dsmInitEx** call. When all threads complete processing, enter a call to **dsmCleanUp**. The primary process should not end before all the threads complete processing. See callmt1.c in the sample application.

Restriction: The default for the API is single-thread mode. If an application does not call **dsmSetUp** with the *mtflag* value set to DSM_MULTITHREAD, the API permits only one session for each process.

For UNIX or Linux for versions 3.1.6 through version 4.1.2, you cannot use the Trusted Communication Agent in multithread mode. If you want to set the ***passwordaccess*** option to ***generate***, you must be an -Authorized user. For version 4.2 and beyond, this is no longer true.

Once **dsmSetUp** successfully completes, the application can begin multiple threads and enter multiple **dsmInitEx** calls. Each **dsmInitEx** call returns a handle for that session. Any subsequent calls on that thread for that session must use that handle value. Certain values are process-wide, environmental variables (values that are set on **dsmSetUp**). Each **dsmInitEx** call parses options again. Each thread can run with different options by specifying an overwrite file or an options string on the **dsmInitEx** call. This enables different threads to go to different servers, or use different node names.

Recommendation: On HP, set the thread stack to 64K or greater. The default value of the thread stack (32K) might not be sufficient

To permit application users to have a LAN-free session, use **dsmSetUp** *mtFlag DSM_MULTITHREAD* in your application. This is necessary even if the application is single threaded. This flag activates the threading necessary for the Tivoli Storage Manager LAN-free interface.

Signals and signal handlers

The application handles signals from the user or the operating system. If the user enters a **CTRL+C** keystroke sequence, the application must catch the signal and send **dsmTerminate** calls for each of the active threads. Then, call **dsmCleanUp** to exit. If sessions are not closed properly, unexpected results might occur on the server.

The application requires signal handlers, such as SIGPIPE and SIGUSR1, for signals that cause the application to end. The application then receives the return code from the API. For example, to ignore SIGPIPE add the following instruction in your application: `signal(SIGPIPE, SIG_IGN)`. After this information is added, instead of the application exiting on a broken pipe, the proper return code is returned.

You can use the child process, Trusted Communication Agent (TCA) if the `passwordaccess` option is set to `generate`. When you use the TCA process, IBM Tivoli Storage Manager uses the SIGCLD signal. If your application uses the SIGCLD signal, be aware of potential interference from Tivoli Storage Manager processes and how SIGCLD is used. For more information about using the TCA, see "Session security" on page 18.

Starting or ending a session

IBM Tivoli Storage Manager is a session-based product, and all activities must be performed within a Tivoli Storage Manager session. To start a session, the application starts the **dsmInitEx** call. This call must be performed before any other API call other than **dsmQueryApiVersionEx**, **dsmQueryCliOptions**, or **dsmSetUp**.

The **dsmQueryCliOptions** function can be called only before the **dsmInitEx**call. The function returns the values of important options, such as option files, compression settings, and communication parameters. The **dsmInitEx** call sets up a session with the server as indicated in the parameters that are passed in the call or defined in the options files.

The client node name, the owner name, and the password parameters are passed to the **dsmInitEx** call. The owner name is case-sensitive, but the node name and password are not. The application client nodes must be registered with the server before a session starts.

Each time an API application client starts a session with the server, the client application type is registered with the server. Always specify an operating system abbreviation for the application type value because this value is entered in the platform field on the server. The maximum string length is DSM_MAX_PLATFORM_LENGTH.

The **dsmInitEx** function call establishes the Tivoli Storage Manager session with the API configuration file and option list of the application client. The application client can use the API configuration file and option list to set a number of Tivoli Storage Manager options. These values override the values that are set in the user configuration files during installation. Users cannot change the options that the administrator defines. If the application client does not have a specific configuration file and option list, you can set both of these parameters to NULL. For more information about configuration files, see the following topic:

"Understanding configuration and options files" on page 1

The **dsmInitEx** function call establishes the Tivoli Storage Manager session, by using parameters that permit extended verification.

Check the **dsmInitEx** function call and the **dsmInitExOut** information return code. The administrator canceled the last session if the return code is okay (RC=ok) and the information return code (infoRC) is DSM_RC_REJECT_LASTSESS_CANCELED. To end the current session immediately, call **dsmTerminate**.

The **dsmQuerySessOptions** call returns the same fields as the **dsmQueryCliOptions** call. The call can be sent only within a session. The values reflect the client options that are valid during that session, from option files, and from any overrides from the **dsmInitEx** call.

After a session starts, the application can send a call to **dsmQuerySessInfo** to determine the server parameters that are set for this session. Items such as the policy domain and transaction limits are returned to the application with this call.

End sessions with a **dsmTerminate** call. Any connection with the server is closed and all resources that are associated with this session are freed.

For an example of starting and ending a session, see the following topic:

Figure 2 on page 20

The example defines a number of global and local variables that are used in calls to **dsmInitEx** and **dsmTerminate**. The **dsmInitEx** call takes a pointer to `dsmHandle` as a parameter, while the **dsmTerminate** call takes the `dsmHandle` as a parameter. The example in Figure 3 on page 20 displays the details of **rcApiOut**. The function **rcApiOut** calls the API function **dsmRCMsg**, which translates a return code into a message. The **rcApiOut** call then prints the message for the user. A version of **rcApiOut** is included in the API sample application. The **dsmApiVersion** function is a type definition that is found in the header file `dsmapitd.h`.

Session security

The IBM Tivoli Storage Manager session-based system has security components that permit applications to start sessions in a secure manner. These security measures prohibit unauthorized access to the server and help to insure system integrity.

Every session that is started with the server must complete a sign-on process, requires a password. When the password is coupled with the node name of the client, it insures proper authorization when connecting to the server. The application client provides this password to the API to start the session.

Two methods of password processing are available: *passwordaccess=prompt* or *passwordaccess=generate*. If you use the *passwordaccess=prompt* option, you must include the password value on each **dsmInitEx** call. Or, you can supply the node name and owner name on the **dsmInitEx** call.

Passwords have expiration times associated with them. If a **dsmInitEx** call fails with a password-expired return code (DSM_RC_REJECT_VERIFIER_EXPIRED), the application client must enter the **dsmChangePW** call using the handle that is returned by **dsmInitEx**. This updates the password before the session can be established successfully. The example in Figure 4 on page 21 demonstrates the procedure to change a password by using **dsmChangePW**. The login owner must use a root user ID or an authorized user ID to change the password.

The second method, *passwordaccess=generate*, encrypts and stores the password value in a file. The node name and owner name cannot be supplied on the **dsmInitEx** call, and the system default values are used. This protects the security of the password file. When the password expires, the `generate` parameter creates a new one and updates the password file automatically.

Tips:

1. If two different physical machines have the same Tivoli Storage Manager node name or multiple paths are defined on one node using several server stanzas, *passwordaccess=generate* might only work for the stanza which is used first after password expiration. During the first client-server contact, the user is

prompted for the same password for each server stanza separately, and for each stanza, a copy of the password is stored separately. When the password expires, a new password is generated for the stanza which connects the first client-server contact. All subsequent attempts to connect via other server stanzas fail, because there is no logical link between their respective copies of the old password, and the updated copy generated by the stanza used first after password expiration. In this case, you must update the passwords prior to expiration or after expiration as a recovery from the situation, as follows:

a. Run **dsmadmc** and update the password on the server.
b. Run **dsmc -servername=stanza1** and use the new password to generate a proper entry.
c. Run **dsmc -servername=stanza2** and use the new password to generate a proper entry.

2. For UNIX or Linux: Only the root user or an authorized user can change the password when using *passwordaccess=prompt*. Only the root user or an authorized user can start the password file when using *passwordaccess=generate*. You can use the Trusted Communication Agent (TCA) child process for password processing. The application should be aware of this because a child process and the SIGCLD signal are used. The TCA is not used in these situations:
   - The *passwordaccess* option is set to *prompt*.
   - The login user is root.
   - The caller of the function must be an authorized user.

   Restriction: The options `users` and `groups` are not recognized.

An application can restrict user access by other means, such as setting access filters.

Applications that use multiple IP connections to a single Tivoli Storage Manager server should use the same node name and Tivoli Storage Manager client password for each session. Follow these steps to enable this support:

1. Define one Tivoli Storage Manager server stanza in the dsm.sys file.
2. For the connections not using the default IP address, specify the option values for *TCPserver* address and *TCPport* on the **dsmInitEx** call.

These values override the IP connection information, but the session still uses the same dsm.sys stanza node and password information.

Note: Nodes in a cluster share a single password.

```
dsmApiVersionEx * apiApplVer;
char           *node;
char           *owner;
char           *pw;
char           *confFile = NULL;
char           *options = NULL;
dsInt16_t      rc = 0;
dsUint32_t     dsmHandle;
dsmInitExIn_t  initIn;
dsmInitExOut_t initOut;
char           *userName;
char           *userNamePswd;

memset(&initIn, 0x00, sizeof(dsmInitExIn_t));
memset(&initOut, 0x00, sizeof(dsmInitExOut_t));
memset(&apiApplVer,0x00,sizeof(dsmapiVersionEx));
apiApplVer.version = DSM_API_VERSION; /* Set the applications compile */
apiApplVer.release = DSM_API_RELEASE;  /* time version.         */
apiApplVer.level   = DSM_API_LEVEL;
apiApplVer.subLevel= DSM_API_SUBLEVEL;

printf("Doing signon for node %s, owner %s, with password %s\n", node,owner,pw);

initIn.stVersion = dsmInitExInVersion;
initIn.dsmApiVersionP = &apiApplVer
initIn.clientNodeNameP = node;
initIn.clientOwnerNameP = owner ;
initIn.clientPasswordP = pw;
initIn.applicationTypeP = "Sample-API AIX";
initIn.configfile = confFile;
initIn.options = options;
initIn.userNameP = userName;
initIn.userPasswordP = userNamePswd;
rc = dsmInitEx(&dsmHandle, &initIn, &initOut);

if (rc == DSM_RC_REJECT_VERIFIER_EXPIRED)
{
   printf("*** Password expired. Select Change Password.\n");
   return(rc);
}
else if (rc)
{
   printf("*** Init failed: ");
   rcApiOut(dsmHandle, rc);   /* Call function to print error message */
   dsmTerminate(dsmHandle);    /* clean up memory blocks */
   return(rc);
}
```

Figure 2. An example of starting and ending a session

```
void rcApiOut (dsUint32_t handle, dsInt16_t rc)
{
     char *msgBuf ;

     if ((msgBuf = (char *)malloc(DSM_MAX_RC_MSG_LENGTH+1)) == NULL)
     {
          printf("Abort:  Not enough memory.\n") ;
          exit(1) ;
      }

      dsmRCMsg(handle, rc, msgBuf);
      printf("
      free(msgBuf) ;
      return;
}
```

Figure 3. Details of rcApiOut

```
printf("Enter your current password:");
gets(current_pw);
printf("Enter your new password:");
gets(new_pw1);
printf("Enter your new password again:");
gets(new_pw2);
/* If new password entries don't match, try again or exit. */
/* If they do match, call dsmChangePW.                      */

rc = dsmChangePW(dsmHandle,current_pw,new_pw1);
if (rc)
{
   printf("*** Password change failed.  Rc =
}
else
{
   printf("*** Your new password has been accepted and updated.\n");
}
return 0;
```

Figure 4. An example of changing a password

Setting the passwordaccess option to generate without TCA

The Trusted Communication Agent (TCA) is a child process that normally controls access to the protected password file. On UNIX and Linux systems, you can log on as an authorized user and set the passwordaccess option to generate without starting the TCA.

Restriction: For version 3.1.6 through version 4.1.2, when you are running in a multithreaded mode and the passwordaccess is set to generate, only the root user, or an authorized user, is permitted access. The TCA child process does not start.

Complete the following steps when you set the passwordaccess to generate without the TCA:

1. Write the application with a call to **dsmSetUp** which passes *argv[0]*. The *argv[0]* contains the name of the application that calls the API. The application is permitted to run an authorized user; however, the administrator must decide on the login name for the authorized user.
2. Set the effective user ID bit (S bit) for the application executable to On. The owner of the application executable file can then become an authorized user and can create a password file, update passwords, and run applications. The owner of the application executable file must be the same as the user ID that runs the program. In the following example, *User* is user1, the name of the application executable file is applA, and user1 has read/write permissions on the /home/user1 directory. The applA executable file has the following permissions:

   ```
   -rwsr-xr-x user1    group1   applA
   ```
3. Instruct the users of the application to use the authorized user name to log in. IBM Tivoli Storage Manager verifies that the login ID matches the application executable owner before it permits access to the protected password file.
4. Set the passworddir option in the dsm.sys file to point to a directory where this user has read/write access. For example, enter the following line in the server stanza of the dsm.sys file:

   ```
   passworddir /home/user1
   ```
5. Create the password file and ensure that the authorized user owns the file.
6. Log on as user1 and run app1A.
7. Call **dsmSetUp** and pass in *argv*.

Creating an administrative user with client owner authority

An administrative user with client owner authority can set parameters on the **dsmInitEx** function call to start sessions. This user can function as an "administrative user" with backup and restore authority for the defined nodes.

To receive client owner authority, complete the following steps:

1. Define the administrative user:

   ```
   REGister Admin admin_name password
   ```

   Where:

   - *admin_name* is the administrative user name.
   - *password* is the admin password.

2. Define the authority level. Users with system or policy authority also have client owner authority.

   ```
   Grant Authority admin_name classes authority node
   ```

   Where:

   - *admin_name* is the administrative user.
   - *classes* is the node.
   - *authority* has one of the following levels of authority:
     - `owner`: full backup and restore authority for the node
     - `node`: single node
     - `domain`: group of nodes

3. Define access to a single node.

   ```
   Register Node   node_name  password  userid
   ```

   Where:

   - *node_name* is the client user node
   - *password* is the client user node password
   - *userid* is the administrative user name

When the application uses the administrative user, the **dsmInitEx** function is called with the `userName` and `userNamePswd` parameters.

```
dsmInitEx
      clientNodeName = NULL
      clientOwnerName = NULL
      clientPassword = NULL
      userName = 'administrative user' name
      userNamePswd = 'administrative user' password
```

You can set the `passwordaccess` option to `generate` or `prompt`. With either parameter, the `userNamePswd` value starts the session. When the session starts, any backup or restore process can occur for that node.

Object names and IDs

The IBM Tivoli Storage Manager server is an object storage server whose primary function is to efficiently store and retrieve named objects. The object ID is unique for each object and remains with the object for the life of the object *except* when you use export or import.

To meet this requirement Tivoli Storage Manager has two main storage areas, database and data storage.

- The database contains all metadata, such as the name or attributes associated with objects.
- The data storage contains the object data. The data storage is actually a storage hierarchy that the system administrator defines. Data are efficiently stored and managed on either online or offline media, depending on cost and access needs.

Each object that is stored on the server has a name associated with it. The client controls the following key components of that name:

- File space name
- High-level name
- Low-level name
- Object type

When making decisions about naming objects for an application, you might need to use an external name for the full object names to the end user. Specifically, the end user might need to specify the object in an Include or Exclude statement when the application is run. The exact syntax of the object name in these statements is platform-dependent. On the Windows operating system, the drive letter associated with the file space rather than the file space name itself is used in the Include or Exclude statement.

The object ID value that was assigned when you created the object might not be the same as when you perform a restore process. Applications should save the object name and then query to obtain the current object ID before doing a restore.

File space name

The file space name is one of the most important storage components. It can be the name of a file system, disk drive, or any other high-level qualifier that groups related data together.

IBM Tivoli Storage Manager uses the file space to identify the file system or disk drive on which the data are located. In this way, actions can be performed on all entities within a file space, such as querying all objects within a specified file space. Because the file space is such an important component of the Tivoli Storage Manager naming convention, you use special calls to register, update, query, and delete file spaces.

The server also has administrative commands to query the file spaces on any node in Tivoli Storage Manager storage, and delete them if necessary. All data stored by the application client must have a file space name associated with it. Select the name carefully to group similar data together in the system.

To avoid possible interference, an application client should select different file space names from those that a backup-archive client would use. The application

client should publish its file space names so that end users can identify the objects for include-exclude statements, if necessary.

Note: On Windows platforms, a drive letter is associated with a file space. When you register or update a file space, you must supply the drive letter. Because the include-exclude list refers to the drive letter, you must keep track of each letter and its associated file space. In the sample program dapismp, the drive letter is set to "G" by default.

See Chapter 2, “Building and running the sample API application,” on page 5 for more information on the sample programs.

High-level and low-level names

Two other components of the object name are the high-level name qualifier and the low-level name qualifier. The high-level name qualifier is the directory path in which the object belongs, and the low-level name qualifier is the actual name of the object in that directory path.

When the file space name, high-level name, and low-level name are concatenated, they must form a syntactically correct name on the operating system on which the client runs. It is not necessary for the name to exist as an object on the system or resemble the actual data on the local file system. However, the name must meet the standard naming rules to be properly processed by the **dsmBindMC** calls. See “Understanding backup and archive objects” on page 41 for naming considerations that are related to policy management.

Object type

The object type identifies the object as either a file or a directory. A file is an object that contains both attributes and binary data, and a directory is an object that contains only attributes.

Table 8 shows what the application client would code is for object names by platform.

Table 8. Application object name examples by platform

Platform	Client code for object name
UNIX or Linux	/myfs/highlev/lowlev
Windows	"myvol\\highlev\\lowlev" **Note:** On a Windows platform, a double backslash translates into a single backslash, because a backslash is the escape character. File space names start with a slash on the UNIX or Linux platform, but do not start with a slash on the Windows platform.

Accessing objects as session owner

Each object has an owner name associated with it. The rules determining what objects are accessed depend on what owner name is used when a session is started. Use this session owner value to control access to the object.

The session owner is set during the call to **dsmInitEx** in the *clientOwnerNameP* parameter. If you start a session with **dsmInitEx** owner name of *NULL* and you use *passwordaccess=prompt*, that session owner is handled with session (root or authorized user) authority. This is also true if you log in with a root user ID or an authorized user ID and you use *passwordaccess= generate*. During a session started in this manner, you can perform any action on any object that is owned by this node regardless of the actual owner of that object.

If a session is started with a specific owner name, the session can only perform actions on objects that have that object owner name associated with them. Backups or archives into the system all must have this owner name associated with them. Any queries performed return only the values that have this owner name associated with them. The object owner value is set during the **dsmSendObj** call in the **Owner** field of the **ObjAttr** structure. An owner name is case-sensitive. Table 9 summarizes the conditions under which a user has access to an object.

Table 9. Summary of user access to objects

Session owner	Object owner	User access
NULL (root, system owner)	" " (empty string)	Yes
NULL	Specific name	Yes
Specific name	" " (empty string)	No
Specific name	Same name	Yes
Specific name	Different name	No

Accessing objects across nodes and owners

Three function calls support cross-node, cross-owner access on the same platform: **dsmSetAccess**, **dsmDeleteAccess**, and **dsmQueryAccess**. These functions, along with the *-fromnode* and *-fromowner* string options that are passed on **dsmInitEx**, permit a complete cross-node query, restore and retrieve process through the API.

For example, User A on node A uses the **dsmSetAccess** function call to give access to its backups under the /db file space to User B from Node B. The access rule is displayed as:

ID	Type	Node	User	Path
1	Backup	Node B	User B	/db/*/*

When User B logs on at Node B, the option string to **dsmInitEx** is:

```
-fromnode=nodeA -fromowner=userA
```

These options are set for this session. Any queries access the file spaces, and files of Node A. Backups and archives are not permitted. Only query, restore, and retrieve processes are permitted from the file spaces for which User B has access. If the application tries to execute any operation using a **dsmBeginTxn** (for examples, backup or update) while signed in with a *-fromnode* or *-fromowner* option set, then

the **dsmBeginTxn** fails with the return code DSM_RC_ABORT_NODE_NOT_AUTHORIZED. See the individual function calls and "**dsmInitEx**" on page 115 for more information.

Tip: On UNIX and Linux you can specify *–fromowner=root* in the option string that is passed on the **dsmInitEx** function call. This permits non-root users access to files that the root owns if a set access was performed.

Use the *asnodename* option on the **dsmInitEx** option string with the appropriate function to back up, archive, restore, retrieve, query or delete data under the target node name on the IBM Tivoli Storage Manager server. See "Backing up multiple nodes with client node proxy support" on page 80 for information on enabling this option.

Managing file spaces

Because file spaces are important to the operation of the system, a separate set of calls is used to register, update, and delete file space identifiers. Before you can store any objects that are associated with a file space on the system, you must first register the file space with IBM Tivoli Storage Manager.

Use the **dsmRegisterFS** call to accomplish this task. For more information about object names and IDs, see "Object names and IDs" on page 23.

The file space identifier is the top-level qualifier in a three-part name hierarchy. Grouping related data together within a file space makes management of that data much easier. For example, either the application client or the Tivoli Storage Manager server administrator can delete a file space and all the objects within that file space.

File spaces also permit the application client to provide information about the file space to the server that the administrator can then query. This information is returned on the query in the **qryRespFSData** structure and includes the following file system information:

Type	Definition
fstype	The file space type. This field is a character string that the application client sets.
fsAttr[platform].fsInfo	A client information field that is used for client-specific data.
capacity	The total amount of space in the file space.
occupancy	The amount of space that is currently occupied in the file space.
backStartDate	The time stamp when the latest backup started (set by sending a **dsmUpdateFS** call).
backCompleteDate	The time stamp when the latest backup completed (set by sending a **dsmUpdateFS** call).

Using capacity and occupancy depends on the application client. Some applications might not need information about the size of the file space, in which case these fields can default to `0`. For more information about querying file spaces, see "Querying the Tivoli Storage Manager system" on page 33.

After a file space is registered with the system, you can back up or archive objects at any time. To update the occupancy and the capacity fields of the file space after

a backup or archive operation, call **dsmUpdateFS**. This call ensures that the values for the occupancy and capacity of the file system are current. You can also update the **fsinfo**, **backupstart**, and **backupcomplete** fields.

If you want to monitor your last backup dates, enter a **dsmUpdateFS** call before you start the backup. Set the update action to DSM_FSUPD_BACKSTARTDATE. This forces the server to set the **backStartDate** field of the file space to the current time. After the backup is complete for that file space, enter a **dsmUpdateFS** call with the update action that is set to DSM_FSUPD_BACKCOMPLETEDATE. This call creates a time stamp on the end of the backup.

If a file space is no longer needed, you can delete it with the **dsmDeleteFS** command. On a UNIX or Linux operating system, only the root user or authorized users can delete file spaces.

The examples in Figure 5 demonstrate how to use the three file space calls for UNIX or Linux. For an example of how to use the three file space calls for Windows, see the sample program code that is installed on your system.

```
/*  Register the file space if it has not already been done.  */

dsInt16        rc;
regFSData    fsData;
char         fsName[DSM_MAX_FSNAME_LENGTH];
char         smpAPI[] = "Sample-API";

strcpy(fsName,"/home/tallan/text");
memset(&fsData,0x00,sizeof(fsData));
fsData.stVersion = regFSDataVersion;
fsData.fsName = fsName;
fsData.fsType = smpAPI;
strcpy(fsData.fsAttr.unixFSAttr.fsInfo,"Sample API FS Info");
fsData.fsAttr.unixFSAttr.fsInfoLength =
       strlen(fsData.fsAttr.unixFSAttr.fsInfo) + 1;
fsData.occupancy.hi=0;
fsData.occupancy.lo=100;
fsData.capacity.hi=0;
fsData.capacity.lo=300;

rc = dsmRegisterFS(dsmHandle,fsData);
if (rc == DSM_RC_FS_ALREADY_REGED) rc = DSM_RC_OK;  /* already done */
if (rc)
{
   printf("Filespace registration failed: ");
   rcApiOut(dsmHandle, rc);
   free(bkup_buff);
   return (RC_SESSION_FAILED);
}
```

Figure 5. An example of working with file spaces, Part 1

```
/* Update the file space. */

dsmFSUpd     updFilespace;         /* for update FS */

updFilespace.stVersion = dsmFSUpdVersion;
updFilespace.fsType = 0;             /* no change */
updFilespace.occupancy.hi = 0;
updFilespace.occupancy.lo = 50;
updFilespace.capacity.hi = 0;
updFilespace.capacity.lo = 200;
strcpy(updFilespace.fsAttr.unixFSAttr.fsInfo,
       "My update for filespace") ;
updFilespace.fsAttr.unixFSAttr.fsInfoLength =
       strlen(updFilespace.fsAttr.unixFSAttr.fsInfo);

updAction = DSM_FSUPD_FSINFO |
            DSM_FSUPD_OCCUPANCY |
            DSM_FSUPD_CAPACITY;

rc = dsmUpdateFS (handle,fsName,&updFilespace,updAction);
printf("dsmUpdateFS rc=%d\n", rc);
```

Figure 6. An example of working with file spaces, Part 2

```
/*  Delete the file space.  */

printf("\nDeleting file space
rc = dsmDeleteFS (dsmHandle,fsName,DSM_REPOS_ALL);
if (rc)
{
   printf("  FAILED!!! ");
   rcApiOut(dsmHandle, rc);
}
else printf("  OK!\n");
```

Figure 7. An example of working with file spaces, Part 3

Associating objects with management classes

A primary feature of IBM Tivoli Storage Manager is the use of policies (management classes) to define how objects are stored and managed in Tivoli Storage Manager storage. An object is associated with a management class when the object is backed up or archived.

This management class determines:
- How many versions of the object are kept if backed up
- How long to keep archive copies
- Where to insert the object in the storage hierarchy on the server

Management classes consist of both backup copy groups and archive copy groups. A copy group is a set of attributes that define the management policies for an object that is being backed up or archived. If a backup operation is being performed, the attributes in the backup copy group apply. If an archive operation is being performed, the attributes in the archive copy group apply.

The backup or archive copy group in a particular management class can be empty or NULL. If an object is bound to the NULL backup copy group, that object cannot be backed up. If an object is bound to the NULL archive copy group, the object cannot be archived.

Because the use of a policy is a very important component of Tivoli Storage Manager, the API requires that all objects sent to the server are first assigned a management class by using the **dsmBindMC** call. With Tivoli Storage Manager software, you can use an include-exclude list to affect management class binding. The **dsmBindMC** call uses the current Include-Exclude list to perform management class binding.

Include statements can associate a specific management class with a backup or archive object. Exclude statements can prevent objects from being backed up but not from being archived.

The API requires that **dsmBindMC** is called before you back up or archive an object. The **dsmBindMC** call returns a mcBindKey structure that contains information on management class and copy groups that are associated with the object. Check the copy group destination before proceeding with a send. When you send multiple objects in a single transaction, they must have the same copy group destination. The **dsmBindMC** function call returns the following information:

Table 10. Information returned on the dsmBindMC call

Information	Description
Management Class	The name of the management class that was bound to the object. The application client can send the **dsmBeginQuery** call to determine all attributes of this management class.
Backup Copy Group	Informs you if a backup copy group exists for this management class. If a backup operation is being performed and a backup copy group does not exist, this object cannot be sent to storage. You receive an error code if you attempted to send it using the **dsmSendObj** call.
Backup Copy Destination	This field identifies the storage pool to which the data is sent. If you are performing a multiple object backup transaction, all copy destinations within that transaction must be the same. If an object has a different copy destination than previous objects in the transaction, end the current transaction and begin a new transaction before you can send the object. You receive an error code if you attempt to send objects to different copy destinations within the same transaction.
Archive Copy Group	Informs you if an archive copy group exists for this management class. If an archive operation is being performed and an archive copy group does not exist, this object cannot be sent to storage. You receive an error code if you attempted to send it using the **dsmSendObj** call.
Archive Copy Destination	This field identifies the storage pool to which the data are sent. If you are performing a multiple object archive transaction, all copy destinations within that transaction must be the same. If an object has a different copy destination than previous objects in the transaction, end the current transaction and begin a new transaction before you send the object. You receive an error code if you attempt to send objects to different copy destinations within the same transaction.

Backup copies of an object can be rebound to a different management class if a subsequent back up with the same object name is done that uses a management class different than the original. For example, if you back up ObjectA and bind it to Mgmtclass1, and later you back up ObjectA and bind it to Mgmtclass2, the most current backup rebinds any inactive copies to Mgmtclass2. The parameters defined in Mgmtclass2 would now control all copies. However the data does not move if the destination is different.

You can also rebind backup copies to a different management class using the **dsmUpdateObj** or **dsmUpdateObjEx** call with the DSM_BACKUPD_MC action.

Related reference:

Deduplication option

Query management classes

Applications can query management classes to determine what management classes are possible for a given node and to determine what the attributes are within the management class.

You can only bind objects to management classes by using the **dsmBindMC** call. You might want your applications to query the management class attributes and display them to end users. See "Querying the Tivoli Storage Manager system" on page 33 for more information.

In the example in Figure 8, a switch statement is used to distinguish between backup and archive operations when calling **dsmBindMC**. The information returned from this call is stored in the **MCBindKey** structure.

```
dsUint16_t     send_type;
dsUint32_t     dsmHandle;
dsmObjName   objName;     /* structure containing the object name  */
mcBindKey    MCBindKey;   /* management class information          */
char         *dest;       /* save destination value                */

switch (send_type)
{
   case (Backup_Send) :
      rc = dsmBindMC(dsmHandle,&objName,stBackup,&MCBindKey);
      dest = MCBindKey.backup_copy_dest;
      break;
   case (Archive_Send) :
      rc = dsmBindMC(dsmHandle,&objName,stArchive,&MCBindKey);
      dest = MCBindKey.archive_copy_dest;
      break;
   default : ;
}

if (rc)
{
   printf("*** dsmBindMC failed: ");
   rcApiOut(dsmHandle, rc);
   rc = (RC_SESSION_FAILED);
   return;
}
```

Figure 8. An example of associating a management class with an object

Expiration/deletion hold and release

You can hold deletion and expiration of specific archive objects in response to a pending or ongoing action that requires that particular data be held. In the event an action is initiated that might require access to data, that data must be available until the action is concluded and access to the data is no longer required as part of that process. After determining that the suspension is no longer required (released), normal deletion and expiration timing resumes per the original retention period.

Verify that the server is licensed by issuing a test **dsmRetentionEvent** call:

1. Query for one object you want to hold and get the ID.
2. Issue the **dsmBeginTxn**, **dsmRetentionEvent** with `Hold`, and **dsmEndTxn**.
3. If the server is not licensed, you receive a vote of abort with reason code `DSM_RC_ABORT_LICENSE_VIOLATION`.

Restrictions:

1. You cannot issue more than one **dsmRetentionEvent** call in a single transaction.
2. You cannot issue a hold on an object that is already under hold.

1. To hold objects, complete the following steps:
   a. Query the server for all the objects that you want to place under hold. Get the object ID for each object.
   b. Issue a **dsmBeginTxn** call, then issue a **dsmRetentionEvent** call with the list of objects, followed by a **dsmEventType**: `eventHoldObj` call. If the number of objects exceeds the value of `maxObjPerTxn`, use multiple transactions.
   c. Use the `qryRespArchiveData` response on the **dsmGetNextQObj** function call to confirm that the objects are put under hold. Check the value of `objHeld` in `qryRespArchiveData`.
2. To release objects from hold, complete the following steps:
   a. Query the server for all the objects that you want to release from hold. Get the object ID for each object.
   b. Issue a **dsmBeginTxn** call, then issue a **dsmRetentionEvent** call with the list of objects, followed by a **dsmEventType**: `eventReleaseObj` call. If the number of objects exceeds the value of `maxObjPerTxn`, use multiple transactions.
   c. Use the **qryRespArchiveData** response on the `dsmGetNextQObj` function call to confirm if the objects were released from hold. Check the value of `objHeld` in **qryRespArchiveData**.

Archive data retention protection

Data that is under the control of IBM Tivoli Storage Manager cannot be modified by unauthorized agents, such as an individual or a program. This protection extends to preventing the deletion of data, such as archive objects, by any agent before the expiration of the retention period.

Protecting archive retention helps to ensure that no individual or program can maliciously or accidentally delete data that is under the control of Tivoli Storage Manager. An archive object that is sent to an archive retention protection server is protected from accidental deletes and has an enforced retention period. Archive retention protection has the following restrictions:

- Only archive operations are allowed on a retention protection server.
- Any object that is not bound explicitly to a management class through a value in the **dsmBindMc** function call or through include-exclude statements is bound to the explicit name of the default management class. For example, if the default management class in the node policy is MC1, the object is bound explicitly to MC1 rather than to DEFAULT. On a query response, the object displays as bound to MC1.
- After you enable archive data retention protection, any attempt to delete an object before the retention period expires returns the code `DSM_RC_ABORT_DELETE_NOT_ALLOWED` on the end transaction.

See the Tivoli Storage Manager server documentation for instructions for setting retention protection for an archive object.

To set up archive data retention protection, complete the following steps:

1. On a new server installation with no previous data, run the **SET ARCHIVERETENTIONPROTECTION ON** command.
2. In the API option string on the **dsmInit** or **dsmInitEx** function calls, enter the following instruction:

   ```
   -ENABLEARCHIVERETENTIONPROTECTION=yes
   ```

You can also set the enablearchiveretentionprotection option in your dsm.opt file on systems other than UNIX, or in your dsm.sys file on UNIX systems:

```
SERVERNAME srvr1.ret
   TCPPORT                          1500
   TCPSERVERADDRESS                 node.domain.company.com
   COMMMETHOD                       TCPIP
   ENABLEARCHIVERETENTIONPROTECTION YES
```

For more information about this option, see "The enablearchiveretentionprotection option."

3. Issue a query to the server to confirm that the Tivoli Storage Manager server is enabled for archive retention protection. Check the value of the archiveRetentionProtection field in the dsmQuerySessInfo structure.

The enablearchiveretentionprotection option

The enablearchiveretentionprotection option specifies whether to enable data retention protection for archive objects on the IBM Tivoli Storage Manager server that is dedicated for this purpose. Your server administrator must activate data retention protection on a new server that does not already have stored objects (backup, archive, or space-managed). If the API application attempts to store a backup version or space-managed object on the server, an error message is issued.

The note in Chapter 3, "Considerations for designing an application," on page 11 states: "Do not store objectID values to use for future restores. They are not guaranteed to be persistent during the life of the object." can be relaxed for Archive manager applications since the archive-manager server does not support export or import. Archive-manager applications can save and use the objectID to improve the performance during object restore.

If the server issues the **SET ARCHIVERETENTIONPROTECTION ON** command, you cannot delete an archived object from the server by using the **delete filespace** command, until the policy parameters of the archive copy group are satisfied. See the appropriate server documentation for information about how to set up a management class.

Event-based retention policy

In an event-based retention policy, the retention time of an archive object is initiated by a business event, such as closing a bank account. Event-based retention closely aligns the IBM Tivoli Storage Manager data retention policy with business requirements for data. When the event occurs, the application sends an **eventRetentionActivate** event for that object to the server to initiate the retention.

To use an event-based retention policy, complete the following steps:

1. On the server, create a management class with an archive **copygroup** of type EVENT. For more information, see the Tivoli Storage Manager server documentation.
2. Query the management class to confirm that the class is event-based. If the management class is event-based, the **retainInit** field in the **archDetailCG** structure is ARCH_RETINIT_EVENT.
3. Bind the objects to the event-based management class by using include, **archmc**, or explicitly through the **mcNameP** attribute in the **ObjAttr** structure on the **dsmSendObj** function call.
4. At the point that you want to start the retention for the object, query the server for all of the objects that are affected. Check to see whether they are in a

PENDING state, and get the object ID. In a pending state, the **retentionInitiated** field in the **qryRespArchiveData** structure indicates DSM_ARCH_RETINIT_PENDING.

5. Issue a **dsmBeginTxn** call, then issue a **dsmRetentionEvent** call with the list of objects, followed by a **dsmEventType**: eventRetentionActivate call. If the number of objects exceeds the value of maxObjPerTxn, use multiple transactions.

   Restriction: You can issue only one **dsmRetentionEvent** call per transaction.

6. Query the objects to confirm that the retention is activated. If retention is initiated, the **retentionInitiated** field in the **qryRespArchiveData** structure has a value of I.

Querying the Tivoli Storage Manager system

The API has several queries, such as management class query, that applications can use.

All queries that use the **dsmBeginQuery** call follow these steps:

1. Send the **dsmBeginQuery** call with the appropriate query type:
   - Backup
   - Archive
   - Active backed-up objects
   - File space
   - Management class

   The **dsmBeginQuery** call informs the API of the data format that is returned from the server. The appropriate fields can be placed in the data structures that are passed by the **dsmGetNextQObj** calls. The begin query call also permits the application client to set the scope of the query by properly specifying the parameters on the begin query call.

   Restriction: On UNIX or Linux systems, only the root user can query active backed-up objects. This query type is known as "fast path".

2. Enter the **dsmGetNextQObj** call to obtain each record from the query. This call passes a buffer that is large enough to hold the data that is returned from the query. Each query type has a corresponding data structure for the data returned. For example, a backup query type has an associated **qryRespBackupData** structure that is populated when the **dsmGetNextQObj** call is sent.
3. The **dsmGetNextQObj** call usually returns one of the following codes:
   - DSM_RC_MORE_DATA: Send the **dsmGetNextQObj** call again.
   - DSM_RC_FINISHED: There is no more data. Send the **dsmEndQuery** call.
4. Send the **dsmEndQuery** call. When all query data are retrieved or more query data are not needed, enter the **dsmEndQuery** call to end the query process. The API flushes any remaining data from the query stream and releases any resources that were used for the query.

Figure 9 on page 34 displays the state diagram for query operations.

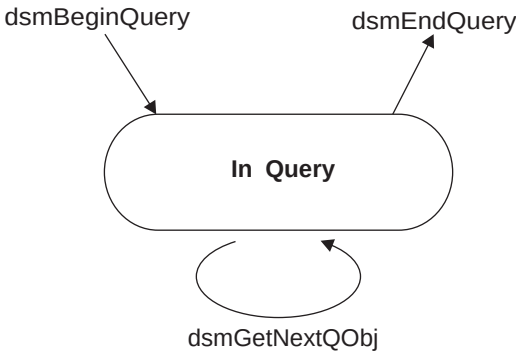


Figure 9. State diagram for general queries

Figure 10 displays the flowchart for query operations.

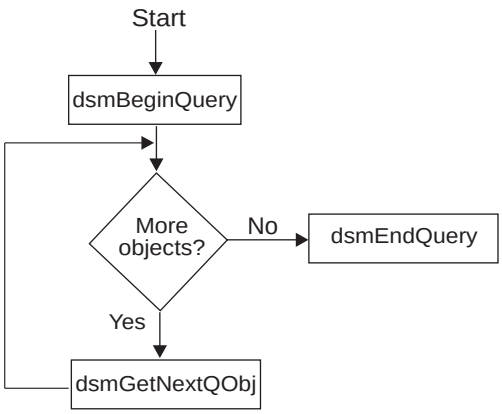


Figure 10. Flowchart for general queries

Example of querying the system

In this example a management class query prints out the values of all the fields in the backup and archive copy groups for a particular management class.

```
dsInt16                rc;
qryMCData            qMCData;
DataBlk              qData;
qryRespMCDetailData  qRespMCData, *mcResp;
char                 *mc, *s;
dsBool_t               done = bFalse;
dsUint32_t               qry_item;

/* Fill in the qMCData structure with the query criteria we want */
qMCData.stVersion = qryMCDataVersion;   /* structure version     */
qMCData.mcName    = mc;                 /* management class name */
qMCData.mcDetail  = bTrue;              /* want full details?    */

/* Set parameters of the data block used to get or send data     */
qData.stVersion = DataBlkVersion;
qData.bufferLen = sizeof(qryRespMCDetailData);
qData.bufferPtr = (char *)&qRespMCData;

qRespMCData.stVersion = qryRespMCDetailDataVersion;
```

```
if ((rc = dsmBeginQuery(dsmHandle,qtMC,(dsmQueryBuff *)&qMCData)))
{
   printf("*** dsmBeginQuery failed: ");
   rcApiOut(dsmHandle, rc);
   rc = (RC_SESSION_FAILED);
}
else
{
   done = bFalse;
   qry_item = 0;
   while (!done)
   {
      rc = dsmGetNextQObj(dsmHandle,&qData);
      if ((   (rc == DSM_RC_MORE_DATA)
           || (rc == DSM_RC_FINISHED))
          && qData.numBytes)
      {
         qry_item++;
         mcResp = (qryRespMCDetailData *)qData.bufferPtr;
         printf("Mgmt. Class
         printf("              Name:
         printf("   Backup CG Name:
            .
            .  /* other fields of backup and archive copy groups */
            .
         printf("   Copy Destination:
      }
      else
      {
         done = bTrue;
         if (rc != DSM_RC_FINISHED)
         {
            printf("*** dsmGetNextQObj failed: ");
            rcApiOut(dsmHandle, rc);
         }
      }
      if (rc == DSM_RC_FINISHED) done = bTrue;
   }
   rc = dsmendQuery (dsmHandle);
}
```

Figure 11. An example of performing a system query

Server efficiency

Use these guidelines when you retrieve from, or send objects to, the Tivoli Storage Manager server.

- When you retrieve objects from the Tivoli Storage Manager server, follow these guidelines:

  - Retrieve data in the restore order that is provided by the Tivoli Storage Manager server. The restore order is especially important for tape devices, because retrieving data that is not ordered can result in tape rewinds and mounts.
  - Even when data is stored on a disk device, you can save time when the retrieves are ordered.
  - Perform as much work as possible in a single Tivoli Storage Manager server session.
  - Do not start and stop multiple sessions.
- When you send objects to the Tivoli Storage Manager server, follow these guidelines:
  - Send multiple objects in a single transaction.
  - Avoid sending one object per transaction, especially when the data is sent directly to a tape device. Part of the tape device transaction is to ensure that the data in the RAM buffers of the tape is written to media.

Related concepts:

"Selecting and sorting objects by restore order" on page 67

Related information:

"Starting or ending a session" on page 17

Sending data to a server

The API permits application clients to send data or named objects and their associated data to IBM Tivoli Storage Manager server storage.

Tip: You can either back up or archive data. Perform all send operations within a transaction.

The transaction model

All data sent to IBM Tivoli Storage Manager storage during a backup or archive operation is done within a transaction. A transaction model provides a high level of data integrity, but it does impose some restrictions that an application client must take into consideration.

Start a transaction by a call to **dsmBeginTxn** or end a transaction by a call to **dsmEndTxn**. A single transaction is an atomic action. Data sent within the boundaries of a transaction is either committed to the system at the end of the transaction or rolled back if the transaction ends prematurely.

Transactions can consist of either single object sends or multiple object sends. To improve system performance by decreasing system overhead, send smaller objects in a multiple object transaction. The application client determines whether single or multiple transactions are appropriate.

Send all objects within a multiple object transaction to the same copy destination. If you need to send an object to a different destination than the previous object, end the current transaction and start a new one. Within the new transaction, you can send the object to the new copy destination.

Note: Objects that do not contain any bit data (*sizeEstimate=0*) are not checked for copy destination consistency.

Tivoli Storage Manager limits the number of objects that can be sent in a multiple object transaction. To find this limit, call **dsmQuerySessInfo** and examine the **maxObjPerTxn** field. This field displays the value of the *TXNGroupmax* option that is set on your server.

The application client must keep track of the objects sent within a transaction to perform retry processing or error processing if the transaction ends prematurely. Either the server or the client can stop a transaction at any time. The application client must be prepared to handle sudden transaction ends that it did not start.

File aggregation

IBM Tivoli Storage Manager servers use a function that is called file aggregation. With file aggregation, all objects sent in a single transaction are stored together, which saves space and improves performance. You can still query and restore the objects separately.

To use this function, all of the objects in a transaction should have the same file space name. If the file space name changes within a transaction, the server closes the existing aggregated object and begins a new one.

LAN-free data transfer

The API can take advantage of LAN-free data transfer if the **dsmSetUp** option for multithreading is ON. The API returns the existence of a LAN-free destination in the **Query Mgmt Class** response structure **archDetailCG** or **backupDetailCG** field **bLanFreeDest**.

You can use LAN-free operations on platforms that are supported by the storage agent. Macintosh platform is excluded.

LAN-free information is provided in the following output structures. The out structure (**dsmEndGetDataExOut_t**) for **dsmEndGetData** includes the field, **totalLFBytesRecv**. This is the total number of LAN-free bytes that are received. The out structure (**dsmEndSendObjExOut_t**) for **dsmEndSendObjEx** includes the field, **totalLFBytesSent**. This is the total number of LAN-free bytes that were sent.

Related information:

LAN-free data movement: Storage agent overview

Simultaneous-write operations

You can configure IBM Tivoli Storage Manager server storage pools to write simultaneously to a primary storage pool and copy storage pool or pools during a backup or archive. Use this configuration to create multiple copies of the object.

If a simultaneous-write operation fails, the return code on the **dsmEndTxn** function might be DSM_RC_ABORT_STGPOOL_COPY_CONT_NO, which indicates that the write to one of the copy storage pools failed, and the Tivoli Storage Manager storage pool option COPYCONTINUE was set to NO. The application terminates and the problem must be resolved by the Tivoli Storage Manager server administrator.

For more information about setting up simultaneous-write operations, see the Tivoli Storage Manager server documentation.

Enhancing API performance

You can use the `tcpbuffsize` and `tcpnodelay` client options and the **DataBlk** API parameter to enhance API performance.

Table 11 describes the actions that you can take to enhance the API performance.

Table 11. Backup-archive options and the API parameter that enhance performance

Backup-archive client options	**Description**
`tcpbuffsize`	Specifies the size of the TCP buffer. The default value is 31 KB. To enhance performance, set the value to 32 KB.
`tcpnodelay`	Specifies whether to send small buffers to the server rather than holding them. To enhance performance, set this option to *yes* for all platforms. This option is valid for Windows and AIX only.
API parameter	**Description**
DataBlk	This parameter is used with the **dsmSendData** function call to determine the application buffer size. For best results, set the parameter as a multiple of the *tcpbuffsize* value that is specified with the `tcpbuffsize` minus 4 bytes. For example, set a value of 28 for this parameter when the value of `tcpbuffsize` is set to 32 KB.

Each **dsmSendData** call is synchronous and does not return until the data transferred to the API in the **dataBlkPtr** is flushed to the network. The API adds a 4-byte overhead to each transaction buffer that is placed on the network.

For example, when the transaction buffer size is 32 KB and the application **DataBlk** buffer size is 31 KB, then each application **DataBlk** buffer fits in a communications buffer and can be flushed immediately. However, if the application **DataBlk** buffer is exactly 32 KB, and because the API is adding 4 bytes per transaction buffer, two flushes are required; one of 32 KB and one of 4 bytes. Also, if you set the `tcpnodelay` option to `no`, flushing the 4 bytes might take up to 200 milliseconds.

Set up the API to send performance data to the client performance monitor

The Tivoli Storage Manager client performance monitor is a component of the Tivoli Storage Manager Administration Center that is used to display performance data that is collected by the Tivoli Storage Manager API. The client performance monitor records and displays performance data for client backup, archive, and restore operations.

With performance monitoring enabled, you can display performance data that is collected by the API by using the performance monitor; the performance monitor is available in the Tivoli Storage Manager Administration Center. Starting with Tivoli Storage Manager Version 7.1, the Administration Center component is no longer included in Tivoli Storage Manager distributions. If you have an Administration Center that was installed with a previous server release, you can continue to use it to display performance data. If you do not already have an Administration Center installed, you can download the previously-released version from ftp://public.dhe.ibm.com/storage/tivoli-storage-management/maintenance/admincenter/v6r3/. For information about using the performance monitor, see the Tivoli Storage Manager Version 6.3 server documentation.

Configuring client performance monitor options

You enable IBM Tivoli Storage Manager clients to use the performance monitor by specifying parameters in the client options file. You specify these options for each client that you want to monitor.

When you monitor performance on UNIX and Linux computers, set the open file descriptor limit to at least 1024, by using the following command:

```
ulimit -n 1024
```

To configure the client performance monitor options, complete the following steps:

1. Open the client options file for each client that you are monitoring. Depending on your configuration, the client options are in one of the following files:
   - `dsm.opt`
   - `dsm.sys`
2. Add the following options to the client options file:

   PERFMONTCPSERVERADDRESS

   PERFMONTCPPORT

   PERFMONCOMMTIMEOUT

PERFMONTCPSERVERADDRESS

The PERFMONTCPSERVERADDRESS option specifies the host name or IP address of the system where the client performance monitor is installed.

Supported clients

This option is platform independent and is supported for all clients.

Options file

Set this option in the client options file (`dsm.opt` or `dsm.sys`).

Syntax

Parameters

server
The server host name or IP address of the system that has the client performance monitor installed (this is the same server that runs the Administration Center).

Examples

Options file:

```
PERFMONTCPSERVERADDRESS 131.222.10.5
```

Command line:

This option cannot be set using the command line.

PERFMONTCPPORT

The port number that the client performance monitor listens on for performance data from the clients.

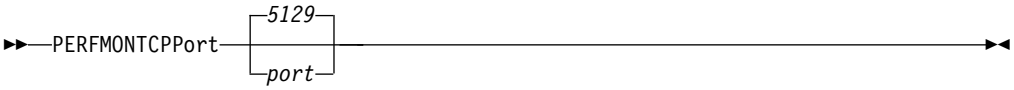
Supported clients

This option is platform independent and is supported for all clients.

Options file

Set this option in the client options file (`dsm.opt` or `dsm.sys`).

Syntax



Parameters

port
The port that is monitored for client performance data. Port 5129 is the default port.

Examples

Options file:

```
PERFMONTCPPPORT 5000
```

Command line:

This option cannot be set using the command line.

PERFMONCOMMTIMEOUT

Specifies the maximum time, in seconds, that the dsmTerminate call waits for performance data to arrive after a session is ended.

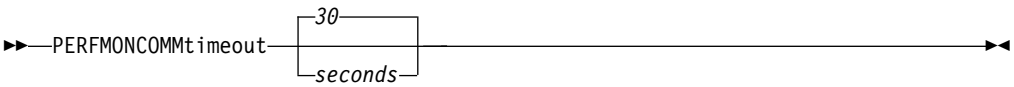
Supported clients

This option is platform independent and is supported for all clients.

Options file

Set this option in the client options file (`dsm.opt` or `dsm.sys`).

Syntax



Parameters

seconds
The time to wait for remaining performance data to arrive, before ending the session.

Examples

Options file:

```
PERFMONCOMMTIMEOUT 60
```

Command line:

This option cannot be set using the command line.

Sending objects to the server

Application clients can send data or named objects and their associated data to IBM Tivoli Storage Manager storage by using the API backup and archive functions. The backup and archive components of the system permit use of different management procedures for data that is sent to storage.

The size estimate attribute is an estimate of the total size of the data object to send to the server. If the application does not know the exact object size, set the *sizeEstimate* to a higher estimate. If the estimate is smaller than the actual size, the Tivoli Storage Manager server uses extra resources to manage extra space allocations.

Tips:

- Be as accurate as is possible when you make this size estimate. The server uses this attribute for efficient space allocation and object placement within its storage resources.
- If the estimate is smaller than the actual size, a server with caching does not allocate extra space and stops the send.

You might encounter problems if the *sizeEstimate* is much too large. The server might not have enough space for the estimated size but does have space for the actual size; or the server might use slower devices.

You can back up or archive objects that are larger than two gigabytes in size. The objects can be either compressed or uncompressed.

To start a send operation, call **dsmSendObj**. If you have more data than you can send at one time, you can make repeated calls to **dsmSendData** to transfer the remainder of the information. Call **dsmEndSendObj** to complete the send operation.

Understanding backup and archive objects

The backup component of the IBM Tivoli Storage Manager system supports several versions of named objects that are stored on the server.

Any object backed up to the server that has the same name as an object that is already stored on the server from that client is subject to version control. Objects are considered to be in active or inactive states on the server. The latest copy of an object on the server that has not been deactivated is in the active state. Any other object with the same name, whether it is an older version or a deactivated copy, is considered inactive. Management class constructs define different management criteria. They are assigned to active and inactive objects on the server.

Table 12 on page 42 lists the copy group fields that apply to active and inactive states:

Table 12. Backup copy group fields

Field	Description
VEREXISTS	The number of inactive versions if active versions exist.
VERDELETED	The number of inactive versions if active versions do not exist.
RETEXTRA	The number of days to keep inactive versions.
RETONLY	The number of days to keep the last inactive versions if active versions do not exist.

If backup versions each have a unique name, such as using a time stamp in the name, then versioning does not happen automatically: every object is active. Active objects never expire, so an application would be responsible for deactivating these with the **dsmDeleteObj** call. In this situation, the application would need the deactivated objects to expire as soon as possible. The user would define a backup copy group with VERDELETED=0 and RETONLY=0.

The archive component of the Tivoli Storage Manager system permits objects to be stored on the server with retention or expiration period controls instead of version control. Each object stored is unique, even though its name might be the same as an object already archived. Archive objects have a description field associated with the metadata that can be used during query to identify a specific object.

Every object on the Tivoli Storage Manager server is assigned a unique object ID. The persistence of the original value is not guaranteed during the life of an object (specifically, after an export or import). Therefore, an application should not query and save the original object ID for use on later restores. Rather, an application should save the object name and insert date. You can use this information during a restore to query objects and verify the insert date. Then, the current object ID can be used to restore the object.

Compression

Configuration options on a given node and the **dsmSendObj** `objCompressed` option, determine whether IBM Tivoli Storage Manager compresses the object during a send. Also, objects with a `sizeEstimate` less than `DSM_MIN_COMPRESS_SIZE` are never compressed.

If the object is compressed already (`objCompressed=bTrue`), it is not compressed again. If it is not compressed, Tivoli Storage Manager decides whether to compress the object, based on the values of the compression option that is set by the administrator and that is set in the API configuration sources.

The administrator can change compression thresholds on the server by using the register node command (`compression=yes`, `no`, or `client-determined`). If this is `client-determined`, then the compression behavior is determined by the compression option value in the configuration sources.

Some types of data, such as data that is already compressed, might actually get bigger when processed with the compression algorithm. When this happens, the return code DSM_RC_COMPRESS_GREW is generated. If you realize that this might happen, but you want the send operation to continue anyway, tell the end users to specify the following option in their options file:

```
COMPRESSAlways Yes
```

If, during a **dsmSendData** function, with compression enabled, you get DSM_RC_COMPRESS_GREW return code, you might want to start over and send the object again without compression. To enforce this, set the **dsmSendObj** `ObjAttr.objCompressed` to `bTrue`.

Information about the actual compression behavior during a **dsmSendObj** is returned by the **dsmEndSendObjEx** call. `objCompressed` specifies if compression was done. `totalBytesSent` is the number of bytes sent by the application. `totalCompressedSize` is the number of bytes after compression. The **dsmEndSendObjEx** call also has a `totalLFBytesSent` field that contains the total bytes sent over LAN-free.

Attention: If your application plans to use partial object restore or retrieve, you cannot compress the data while sending it. To enforce this, set the **dsmSendObj** `ObjAttr.objCompressed` to `bTrue`.

Compression type

The type of compression that the client uses is determined by the combination of compression and client-side data deduplication that is used during backup or archive processing.

The compression algorithm that is used by the client is reported by the API in a new field in the **qryRespArchiveData** and **qryRespBackupData** structures:

```
dsChar_t        compressAlg[20];  /* compression algorithm name */
```

The following types of compression are reported:

LZ4 A faster and more efficient compression method that the client uses when a client-deduplicated object is sent to an LZ4-compatible container storage pool on the Tivoli Storage Manager server. The server must be at version 7.1.5 or later, and must use container storage pools. Client-side LZ4 compression is used only when client-side data deduplication is enabled.

LZW A traditional type of compression that the client uses in any of the following situations:

- Client-deduplicated objects are sent to traditional (non-container) storage pools on the server.
- The client object does not undergo client-side data deduplication.
- The client object undergoes only traditional server-side data deduplication.

Blank field
The object is not compressed by the client. The object is not compressed because the `compression` option is set to *no*, or the option is not specified during backup or archive processing. Although the object is not compressed by the client, it might be compressed by the server.

The compression type is not configurable. It is determined by the backup-archive client at the time of backup or archive processing.

Example

The following example shows the `Compresssion Type` field in the output of the backup and archive queries from the 64-bit sample application **dapismp**:

```
Enter selection ==>1
                      Filespace:\fs1
                      Highlevel:\hl
```

```
                          Lowlevel:\ll
                 Object Type(D/F/A):f
   Active(A),Inactive(I),Both(B):a
If root, query all owners? (Y/N):
                Object Owner Name:
   point in time date (MMDDYYYY):
   point in time time     (hhmm):
     Show detailed output? (Y/N):y
On Restore, Wait for mount?(Y/N):
Are the above responses correct (y/n/q)?


Item 1: \fs1\hl\ll
   Object type: File
   Object state: Active
   Insert date: 2016/2/3 10:57:41
   Expiration date: 0/0/0 0:0:0
   Owner:
   Restore order: 0-0-0-0-0
   Object id: 0-40967
   Copy group: 1
   Media class: Fixed
   Mgmt class: DEFAULT
   Object info is        :Tivoli Storage Manager API Verify Data
   Object info length is :73
   Estimated size : 0  4000
   Compression : YES
   Compression Type: LZ4
   Encryption : NO
   Encryption Strength : NONE
   Client Deduplicated : YES
```

Buffer copy elimination

The buffer copy elimination function removes the copy of data buffers between an application and the IBM Tivoli Storage Manager server, which results in better processor utilization. For maximum effect, use this approach in a LAN-free environment.

The buffers for data movement are allocated by Tivoli Storage Manager and a pointer is passed back to the application. The application places the data in the provided buffer, and that buffer is passed through the communication layers to the storage agent by using shared memory. The data is then moved to the tape device, which eliminates copies of data. This function can be used with either backup or archive operations.

Attention: When you use this method, pay extra attention to proper buffer handling and sizes of buffers. The buffers are shared between the components and any memory overwrite that is a result of a programming error results in severe errors.

The overall sequence of calls for backup/archive is as follows:

```
dsmInitEx (UseTsmBuffers = True, numTsmBuffers = [how many Tivoli Storage Manager
            -allocated buffers the application needs to allocate])
dsmBeginTxn
for each object in the txn
  dsmBindMC
    dsmSendObject
      dsmRequestBuffer
      dsmSendBufferData (sends and release the buffer used)
    dsmEndSendObjEx
```

```
dsmEndTxn
for each buffer still held
      dsmReleaseBuffer
dsmTerminate
```

The **dsmRequestBuffer** function can be called multiple times, up to the value that is specified by the numTsmBuffers option. An application can have two threads: a producer thread that fills buffers with data; and a consumer thread that sends those buffers to Tivoli Storage Manager with the **dsmSendBufferData** call. When a **dsmRequestBuffer** call is issued and the **numTsmBuffers** is reached, the **dsmRequestBuffer** call blocks until a buffer is released. The buffer release can happen by either calling **dsmSendBufferData**, which sends and releases a buffer or by calling **dsmReleaseBuffer**. For more information, see callbuff.c in the API sample directory.

If at any point there is a failure in the send, the application must release all the buffers that are held and terminate the session. For example:

```
If failure
  for each TSM buffer held by application
    call dsmReleaseBuffer
dsmTerminate
```

If an application calls **dsmTerminate** and a buffer is still held, the API does not exit. The following code is returned: DSM_RC_CANNOT_EXIT_MUST_RELEASE_BUFFER. If the application cannot release the buffer, the application must exit the process to force a cleanup.

Buffer copy elimination and restore and retrieve

The IBM Tivoli Storage Manager server controls the amount of data to be placed in the buffer, based on tape access optimization with restore and retrieve. This method is not as beneficial to the application as the normal method of getting data. During prototyping, check the performance of the buffer copy elimination method and use this method only if you see a worthwhile improvement.

The maximum amount of data in a single buffer that is returned by the Tivoli Storage Manager server is (256K bytes – header overhead). As a consequence, only applications that deal with small buffer writes benefit from this data retrieval mechanism. The application must give special attention to the number of bytes in the buffer, depending on the object size, the network, and other boundary conditions. In some situations, the use of buffer copy elimination can actually perform worse than the normal restore. The API normally caches the data and returns a fixed length to the application. The application can then control the number of data writes back to the disk.

If you use buffer copy elimination, create a data-caching mechanism for buffers that are less than the preferred write buffer size. For example, if an application writes 64K data blocks to disk, the application must take these actions:

1. Call **dsmGetBufferData**.
2. Write out blocks of 64K.
3. On the final block, copy the remainder to a **tempBuff**, issue another **dsmGetBufferData** call, and fill the **tempBuff** with the rest of the data.
4. Continue writing blocks of 64K:

```
dsmGetBufferData #1 get 226K        dsmGetBufferData #2 get 240K
Block1 64K - write to disk          Block1 30K - copy to tempbuff-write to disk
Block2 64K - write to disk          Block2 64K - write to disk
Block3 64K - write to disk          Block3 64K - write to disk
Block4 34K - copy to tempbuff       Block4 64K - write to disk
Block5 18K - write to tempbuff        etc
```

In this example, six disk writes are direct and 1 is cached.

The overall sequence of calls for restore and retrieve is as follows:

dsmInitEx (UseTsmBuffers = True numTsmBuffers = how many buffers the application wants to allocate).

```
dsmBeginGetData
While obj id
  dsmGetObj (no data restored on this call- buffer set to NULL)
 While data to read
   dsmGetBufferData (returns the data in the TSM buffer)
  ...process data...
  dsmReleaseBuffer
dsmEndGetObj
dsmEndGetData
```

For every **dsmGetBufferData** call, implement a **dsmReleaseBuffer** call. The **dsmGetBufferData** and corresponding **dsmReleaseBuffer** do not need to be consecutive. An application might issue multiple **dsmGetBufferData** calls first to get several buffers, and then issue the corresponding **dsmReleaseBuffer** calls later. For sample code that uses this function, see `callbuff.c` in the API sample directory.

Restriction: Because the API provides the buffer and the goal is to minimize processor utilization, more processing of the data in the buffer is not permitted. The application cannot use encryption and compression with buffer copy elimination because both of these operations require data processing and copies.

Implement both the regular data movement path and the buffer copy elimination to enable the user to switch between both paths, based on their needs. If the user must compress or encrypt data, then use the existing mechanism. If there is a processor constraint, then use the new mechanism. Both of these mechanisms are complementary and do not completely replace each other.

API encryption

Two methods are available to encrypt data: application-managed encryption and Tivoli Storage Manager client encryption.

Select and use only one of these methods to encrypt data. The methods are mutually exclusive and if you encrypt data by using both methods, you will be unable to restore or retrieve some data. For example, assume that an application uses application-managed encryption to encrypt object A, and then uses Tivoli Storage Manager client encryption to encrypt object B. During a restore operation, if the application sets the option to use Tivoli Storage Manager client encryption and it tries to restore both objects, only object B can be restored; object A cannot be restored because it was encrypted by the application, not by the client.

Regardless of the encryption method that is used, the Tivoli Storage Manager must enable password authentication. By default, the server uses `SET AUTHENTICATION ON`.

The API uses either AES 128-bit or AES 256-bit encryption. AES 256-bit data encryption provides a higher level of data encryption than AES 128-bit data encryption. Files that are backed up by using AES 256-bit encryption cannot be restored with an earlier client. Encryption can be enabled with or without compression. If you use encryption, you cannot use the partial object restore and retrieve and buffer copy elimination functions.

Application-managed encryption

With application-managed encryption, the application provides the key password to the API (using key DSM_ENCRYPT_USER) and it is the application's responsibility to manage the key password.

Attention: If the encryption key is not saved, and you forgot the key, your data is unrecoverable.

The application provides the key password in the **dsmInitEx** call and must provide the proper key password at restore time.

Attention: If the key password is lost, there is no way to restore the data.

The same key password must be used for backup and restore (or archive and retrieve) operations for the same object. This method does not have a dependency on the IBM Tivoli Storage Manager server level. To set up this method, the application needs to follow these steps:

1. Set the `bEncryptKeyEnabled` variable to `bTrue` in the call to **dsmInitEx**, and set the `encryptionPasswordP` variable to point to a string with the encrypt key password.
2. Set the `include.encrypt` for the objects to `encrypt`. For example, to encrypt all data, set:

   ```
   include.encrypt /.../* (UNIX)
   ```

   and

   ```
   include.encrypt *\...\* (Windows)
   ```

   To encrypt the object `/FS1/DB2/FULL`, set:

   ```
   include.encrypt /FS1/DB2/FULL
   ```
3. Set `ENCRYPTKEY=PROMPT|SAVE` in the option string that is passed to the API in the **dsmInitEx** call on Windows. This option can also be set in `dsm.opt` (Windows) or `dsm.sys` (UNIX or Linux).

By default, the `encryptkey` option is set to `prompt`. This setting ensures that the key does not get stored automatically. If `encryptkey` save is specified, the key is stored by Tivoli Storage Manager on the local machine but then only one key can be valid for all Tivoli Storage Manager operations with the same node name.

After a send of an object, the **dsmEndSendObjEx** specifies whether an object was encrypted and which method was used. Possible values in the *encryptionType* field:

- `DSM_ENCRYPT_NO`
- `DSM_ENCRYPT_USER`
- `DSM_ENCRYPT_CLIENTENCRKEY`

The following table lists the API encryption types, prerequisites, and the functions that are available.

Table 13. API encryption types, prerequisites, and functions available

Type	Prerequisite	Function available
ENCRYPTIONTYPE	None	Set the ENCRYPTIONTYPE in the option string that is passed to the API in the **dsmInitEx** call on Windows. ENCRYPTIONTYPE=AES128 by default.
EncryptKey=save	None	API and backup-archive
EncryptKey=prompt	None	API and backup-archive
EncryptKey=generate	None	API and backup-archive
EnableClientEncryptKey	None	API only

Note: It is advised that the server has authentication turned ON. If authentication is turned OFF, the key is not encrypted, but the data is still encrypted. However, this is not recommended.

Table 14 shows how both Authorized Users and non-Authorized Users can encrypt or decrypt data during a backup or restore operation, depending on the value that is specified for the passwordaccess option. The TSM.PWD file must exist to perform the following authorized-user and non-authorized-user operations. The authorized user creates the TSM.PWD file and sets the encryptkey option to save and the passwordaccess option to generate.

Table 14. Encrypting or decrypting data with application managed key on UNIX or Linux

Operation	passwordaccess option	encryptkey option	Result
Authorized user backup	generate	save	Data encrypted.
	generate	prompt	Data encrypted if encryptionPasswordP contains an encryption password.
	prompt	save	Data encrypted if encryptionPasswordP contains an encryption password.
	prompt	prompt	Data encrypted if encryptionPasswordP contains an encryption password.
Authorized user restore	generate	save	Data encrypted.
	generate	prompt	Data encrypted if encryptionPasswordP contains an encryption password.
	prompt	save	Data encrypted if encryptionPasswordP contains an encryption password.
	prompt	prompt	Data encrypted if encryptionPasswordP contains an encryption password.
Non-authorized user backup	generate	save	Data encrypted.
	generate	prompt	Data encrypted if encryptionPasswordP contains an encryption password.
	prompt	save	Data encrypted if encryptionPasswordP contains an encryption password.
	prompt	prompt	Data encrypted if encryptionPasswordP contains an encryption password.
Non-authorized user restore	generate	save	Data encrypted.
	generate	prompt	Data encrypted if encryptionPasswordP contains an encryption password.
	prompt	save	data encrypted if encryptionPasswordP contains an encryption password.
	prompt	prompt	Data encrypted if encryptionPasswordP contains an encryption password.

Tivoli Storage Manager client encryption

IBM Tivoli Storage Manager client encryption uses the key that is managed by the DSM_ENCRYPT_CLIENTENCRKEY value to protect your data. Client encryption is transparent to the application that is using the API, with the exception that partial object restore operations and retrieve operations are not possible for objects that were encrypted or compressed.

For both Tivoli Storage Manager client encryption and application-managed encryption, the encryption password refers to a string value that is used to generate the actual encryption key. The value for the encryption password option is 1-63 characters in length, but the key that is generated from it is always 8 bytes for 56 DES, 16 bytes for 128 AES, and 32 bytes for 256 AES.

Attention: If the encryption key is not available, data cannot be restored or retrieved. When you use `ENABLECLIENTENCRYPTKEY` for encryption, the encryption key is stored on the server database. For objects that use this method, the server database must exist and have the proper values for the objects for a proper restore. Ensure that you back up the server database frequently to prevent data loss.

This is the simpler method to implement, where one random encryption key is generated per session and it is stored on the Tivoli Storage Manager server with the object in the server database. During restore, the stored key is used for decryption. Using this method, the management of the key is the responsibility of Tivoli Storage Manager, and the application does not have to deal with the key at all. Because the key is stored in the server database, you must have a valid Tivoli Storage Manager database for a restore operation of an encrypted object. When the key is transmitted between the API and the server, it is also encrypted. The transmission of the key is secure, and when the key is stored in the Tivoli Storage Manager server database it is encrypted. The only time that the key is placed in the clear with the export data stream is when a node's data are exported between servers.

To enable Tivoli Storage Manager client encryption, complete the following steps:

1. Specify `-ENABLECLIENTENCRYPTKEY=YES` in the option string that is passed to the API on the dsmInitEx call or set the option in the system option file `dsm.opt` (Windows) or `dsm.sys` (UNIX or Linux).
2. Set the `include.encrypt` for the objects to encrypt. For example, to encrypt all data, set:

   ```
   include.encrypt /.../* (UNIX)
   ```

   and

   ```
   include.encrypt *\...\* (Windows)
   ```

   To encrypt the object `/FS1/DB2/FULL`, set:

   ```
   include.encrypt /FS1/DB2/FULL
   ```

Data deduplication

Data deduplication is a method of reducing storage needs by eliminating redundant data.

Overview

Two types of data deduplication are available on Tivoli Storage Manager: *client-side data deduplication* and *server-side data deduplication*.

Client-side data deduplication is a data deduplication technique that is used on the backup-archive client to remove redundant data during backup and archive processing before the data is transferred to the Tivoli Storage Manager server. Using client-side data deduplication can reduce the amount of data that is sent over a local area network.

Server-side data deduplication is a data deduplication technique that is done by the server. The Tivoli Storage Manager administrator can specify the data deduplication location (client or server) to use with the **DEDUP** parameter on the **REGISTER NODE** or **UPDATE NODE** server command.

Enhancements

With client-side data deduplication, you can:

- Exclude specific files on a client from data deduplication.
- Enable a data deduplication cache that reduces network traffic between the client and the server. The cache contains extents that were sent to the server in previous incremental backup operations. Instead of querying the server for the existence of an extent, the client queries its cache.

  Specify a size and location for a client cache. If an inconsistency between the server and the local cache is detected, the local cache is removed and repopulated.

  Note: For applications that use the Tivoli Storage Manager API, the data deduplication cache must not be used because of the potential for backup failures caused by the cache being out of sync with the Tivoli Storage Manager server. If multiple, concurrent backup-archive client sessions are configured, there must be a separate cache configured for each session.
- Enable both client-side data deduplication and compression to reduce the amount of data that is stored by the server. Each extent is compressed before it is sent to the server. The trade-off is between storage savings and the processing power that is required to compress client data. In general, if you compress and deduplicate data on the client system, you are using approximately twice as much processing power as data deduplication alone.

  The server can work with deduplicated, compressed data. In addition, backup-archive clients earlier than V6.2 can restore deduplicated, compressed data.

Client-side data deduplication uses the following process:

- The client creates extents. *Extents* are parts of files that are compared with other file extents to identify duplicates.
- The client and server work together to identify duplicate extents. The client sends non-duplicate extents to the server.

- Subsequent client data-deduplication operations create new extents. Some or all of those extents might match the extents that were created in previous data-deduplication operations and sent to the server. Matching extents are not sent to the server again.

Benefits

Client-side data deduplication provides several advantages:

- It can reduce the amount of data that is sent over the local area network (LAN).
- The processing power that is required to identify duplicate data is offloaded from the server to client nodes. Server-side data deduplication is always enabled for deduplication-enabled storage pools. However, files that are in the deduplication-enabled storage pools and that were deduplicated by the client, do not require additional processing.
- The processing power that is required to remove duplicate data on the server is eliminated, allowing space savings on the server to occur immediately.

Client-side data deduplication has a possible disadvantage. The server does not have whole copies of client files *until* you back up the primary storage pools that contain client extents to a non-deduplicated copy storage pool. (*Extents* are parts of a file that are created during the data-deduplication process.) During storage pool backup to a non-deduplicated storage pool, client extents are reassembled into contiguous files.

By default, primary sequential-access storage pools that are set up for data deduplication must be backed up to non-deduplicated copy storage pools before they can be reclaimed and before duplicate data can be removed. The default ensures that the server has copies of whole files at all times, in either a primary storage pool or a copy storage pool.

Important: For further data reduction, you can enable client-side data deduplication and compression together. Each extent is compressed before it is sent to the server. Compression saves space, but it increases the processing time on the client workstation.

The following options pertain to data deduplication:

- Deduplication
- Dedupcachepath
- Dedupcachesize
- Enablededupcache
- Exclude.dedup
- Include.dedup

API client-side data deduplication

Client-side data deduplication is used by the API on the backup-archive client, to remove redundant data during backup and archive processing before the data is transferred to the IBM Tivoli Storage Manager server.

Client-side data deduplication is used by the API, to remove redundant data during backup and archive processing before the data is transferred to the Tivoli Storage Manager server. Using client-side data deduplication can reduce the amount of data that is sent over a local area network. Using client-side data deduplication can also reduce the Tivoli Storage Manager server storage space.

When the client is enabled for client-side data deduplication, and you perform a backup or archive operation, the data is sent to the server as extents. The next time a backup or archive operation is performed, the client and server identify which data extents have already been backed up or archived, and send only the unique extents of data to the server.

For client-side data deduplication, the Tivoli Storage Manager server and API must be at Version 6.2 or later.

Before you use client-side data deduplication to back up or archive your files, the system must meet the following requirements:

- The client must have the `deduplication` option enabled.
- The server must enable the client for client-side data deduplication with the **DEDUP=CLIENTORSERVER** parameter on either the **REGISTER NODE** or **UPDATE NODE** command.
- The storage pool destination for the data must be a data deduplication-enabled storage pool. The data deduplication-enabled storage pool is file device type only.
- Ensure that the files are bound to the correct management class.
- A file can be excluded from client-side data deduplication processing. By default, all files are included.
- Files must be larger than 2 KB.
- The server can limit the maximum transaction size for data deduplication by setting the `CLIENTDEDUPTXNLIMIT` option on the server. See the server documentation information about this option.

If any of these requirements are not met, data is processed normally, with no client-side data deduplication.

Here are some data deduplication restrictions:

- LAN-free data movement and client-side data deduplication are mutually exclusive. If you enable both LAN-free data movement and client-side data deduplication, LAN-free data movement operations complete and client-side data deduplication is ignored.
- Encryption and client-side data deduplication are mutually exclusive. If you enable both encryption and client-side data deduplication, encryption operations complete and client-side data deduplication is ignored. Encrypted files, and files that are eligible for client-side data deduplication, can be processed in the same operation, but are done in separate transactions.

Requirements:

1. In any transaction, all files must be either included for data deduplication or excluded. If the transaction has mixed files, the transaction fails, and a return code of DSM_RC_NEEDTO_ENDTXN is returned by the API.
2. Use storage device encryption together with client-side data deduplication. Because SSL is used in combination with client-side deduplication, there is no need for client encryption.

- The following functions are not available for client-side data deduplication:
  - Hierarchical Storage Manager (HSM) client
  - API shared buffer
  - NAS
  - Subfile backup
- Buffer copy elimination cannot be used with data transformations like compression, encryption, and data deduplication.
- If you use client-side deduplication, the API detects and fails (with RC=254) backups of file extents that are marked as expired on the server during sending data to the server. If you want to retry the operation, you need to include that programming in the calling application.
- Simultaneous-write operations on the server takes precedence over client-side data deduplication. If simultaneous-write operations are enabled, client-side data deduplication does not occur.

  Restriction: When client side data deduplication is enabled, the API cannot recover from a state where the server has run out of storage on the destination pool, even if there is a next pool defined. A stop reason code of DSM_RS_ABORT_DESTINATION_POOL_CHANGED is returned and the operation fails. There are two ways to recover from this situation:
  1. Ask the administrator to add more scratch volumes to the original filepool.
  2. Retry the operation with data deduplication disabled.

For even greater bandwidth savings, you can enable a local cache for data deduplication. The local cache saves queries from going to the Tivoli Storage Manager server. The default value for ENABLEDEDUPCACHE is NO, so that the cache is not out of sync with the server. If the cache is out of sync with the server, the application resends all data. If your application can retry on a failed transaction, and you want to use the local cache, set the ENABLEDEDUPCACHE option to YES in the dsm.opt (Windows) or dsm.sys (UNIX) file.

At the end of a restore, if *all* of the data was restored through the API, and the object was deduplicated by the client, an end-to-end digest is calculated and compared to the value calculated at backup time. If those values do not match, error DSM_RC_DIGEST_VALIDATION_ERROR is returned. If an application receives this error, the data is corrupt. This error can also be a result of a transient error on the network, so try the restore or retrieve again.

Here is an example of the query session command showing data deduplication information:

```
dsmQuerySessInfo Values:
Server Information:
Server name: SERVER1
Server Host: AVI
Server port: 1500
Server date: 2009/10/6 20:48:51
Server type: Windows
Server version: 6.2.0.0
Server Archive Retention Protection : NO
Client Information:
```

```
Client node type: API Test1
Client filespace delimiter: :
Client hl & ll delimiter: \
Client compression: Client determined (3u)
Client archive delete: Client can delete archived objects
Client backup delete: Client CANNOT delete backup objects
Maximum objects in multiple object transactions: 4096
Lan free Enabled: NO
Deduplication : Client Or Server
General session info:
Node: AVI
Owner:
API Config file:
```

Here is an example of the query management class command showing data deduplication information:

```
Policy Information:
Domain name: DEDUP
Policyset name: DEDUP
Policy activation date: 0/0/0 0:0:0
Default management class: DEDUP
Backup retention grace period: 30 days
Archive retention grace period: 365 days
Mgmt. Class 1:
Name: DEDUP
Description: dedup - values like standard
Backup CG Name: STANDARD
Frequency: 0
Ver. Data Exists: 2
Ver. Data Deleted: 1
Retain Extra Ver: 30
Retain Only Ver: 60
Copy Destination: AVIFILEPOOL
Lan free Destination: NO
Deduplicate Data: YES

Archive CG Name: STANDARD
Frequency: 10000
Retain versions: 365
Copy Destination: AVIFILEPOOL
Lan free Destination: NO
Retain Init : CREATE
Retain Minimum : 65534
Deduplicate Data: YES
```

Related reference:

Deduplication option

Exclude files from data deduplication

You can choose to exclude backup or archive files from data deduplication.

To exclude files from data deduplication processing, follow these steps:

1. Set the `exclude.dedup` option for the objects to exclude.

   For example, to exclude all dedup data for UNIX systems, set:

   ```
   exclude.dedup /.../*
   ```

2. To exclude all dedup data for Windows systems, set:

   ```
   exclude.dedup *\...\*
   ```

Important: If an object is sent to a data deduplication pool, data deduplication occurs on the server, even if the object is excluded from client-side data deduplication.

Include files for data deduplication

You can choose to include backup or archive files for data deduplication.

To refine the list of files to be included, the `include.dedup` option can be used in combination with the `exclude.dedup` option.

By default, all eligible objects are included for data deduplication.

Here are some UNIX and Linux examples:

```
exclude.dedup /FS1/.../*

include.dedup /FS1/archive/*
```

Here are some Windows examples:

```
exclude.dedup E:\myfiles\...\*

include.dedup E:\myfiles\archive\*
```

Server-side data deduplication

Server-side data deduplication is data deduplication that is performed by the server.

The IBM Tivoli Storage Manager administrator can specify the data deduplication location (client or server) to use with the **DEDUP** parameter on the **REGISTER NODE** or **UPDATE NODE** server command.

In a data deduplication-enabled storage pool (file pool), only one instance of a data extent is retained. Other instances of the same data extent are replaced with a pointer to the retained instance.

For more information about server-side data deduplication, see the Tivoli Storage Manager server documentation.

Application failover

When the IBM Tivoli Storage Manager server becomes unavailable because of an outage, applications that use the API can automatically fail over to a secondary server for data recovery.

The Tivoli Storage Manager server that the client and API connects to during normal production processes is called the *primary server*. When the primary server is set up for node replication, that server is also known as the *source replication server*. The client node data on the source replication server can be replicated to the *target replication server*. This server is also known as the *secondary server*, and is the server that the client automatically fails over to when the primary server fails.

The client and API must be configured for automated client failover, and must connect to a Tivoli Storage Manager V7.1 (or later) server that replicates client node data. The configuration for the API is the same as the configuration for the backup-archive client.

During normal operations, connection information for the secondary server is automatically sent to the client from the primary server during the logon process. The secondary server information is automatically saved to the client options file.

Each time the client application logs on to the Tivoli Storage Manager server, it attempts to contact the primary server. If the primary server is unavailable, the application automatically fails over to the secondary server by using the secondary server information in the client options file. In failover mode, the application can query the secondary server and restore or retrieve replicated data.

You must back up the application at least one time to the primary server. The API can fail over to the secondary server to recover data only if the data from the client node was replicated from the primary server to the secondary server.

Related concepts:

Automated client failover configuration and use

Failover status information

The API provides status information that applications can use to determine the failover status and the status of replicated client data on the secondary server.

The replication status indicates whether the most recent backup was replicated to the secondary server. If the time stamp of the most recent backup operation on the API matches the time stamp of the backup on the secondary server, the replication status is current. If the two time stamps do not match, the replication status is not current and the replicated data might be out-of-date.

The following replication status information is returned on the **query filespace** response on the **dsmGetNextQObj** function call in the **qryRespFSData** structure:

Table 15. Replication status information reported by the API

Status information	Type	Definition
Start of last replication	**lastReplStartDate**	The last time replication was started.
End of last replication	**lastReplCmpltDate**	The last time replication was completed, even if there was a failure.
Last backup store date (Server)	**lastBackOpDateFromServer**	The last store time stamp that was saved on the server.
Last backup store date (Local)	**lastBackOpDateFromLocal**	The last store time stamp that was saved on the client.

The failover status is reported by the **bIsFailOverMode** field in the **dsmInitExOut_t** structure.

See Appendix B, “API type definitions source files,” on page 155 for the structure and type definitions of the API.

The DSM_RC_SIGNON_FAILOVER_MODE return code indicates that the client and API failed over to the secondary server, and is running in failover mode.

Example of signon during a failover

The following sample output is an example of signing on the server during a failover:

```
signon
Doing signon for node khoyt, owner , with password khoytpass
ANS2106I Connection to primary TSM server 123.45.6.78 failed

ANS2107I Attempting to connect to secondary server TARGET at 123.45.6.79 : 1501

ANS2108I Connected to secondary server TARGET.
Handle on return = 1

************************************************************
After dsmInitEx:
Server TARGET ver/rel/lev 7/1/0/0
userNameAuthorities      : Owner
Replication Server name  : TARGET
Home Server name         : MINE
Connected to replication server
************************************************************
```

Example of query session command

The following sample output is an example of the **query session** command that shows the secondary (replication) server information:

```
query session
dsmQuerySessInfo Values:
  Server Information:
    Server name   : TARGET
    Server Host   : 123.45.6.79
    Server port   : 1500
    Server date   : 2013/5/21  14:13:32
    Server type   : Windows
    Server version: 7.1.0.0
    Server Archive Retention Protection : NO
  Replication Server Infomation
    Home Server name        : MINE
    Replication Server name : TARGET
       Host                 : 123.45.6.79
       Port                 : 1501
    Fail over status        : Connected to replication server
  Client Information:
    Client node type          : Unix
    Client filespace delimiter: /
    Client hl & ll delimiter  : /
    Client compression        : Client determined (3u)
    Client archive delete     : Client can delete archived objects
    Client backup delete      : Client CANNOT delete backup objects
    Maximum objects in multiple object transactions: 4096
    Lan free Enabled          : NO
    Deduplication             : Server Only
  General session info:
    Node           : KHOYT
    Access Node    :
    Owner          :
    API Config file:
  Policy Information:
    Domain name                   : STANDARD
    Policyset name                : STANDARD
    Policy activation date        : 0/0/0  0:0:0
    Default management class      : STANDARD
    Backup retention grace period : 30 days
    Archive retention grace period: 365 days
```

Example of query filespace command

The following sample output is an example of the **query filespace** command that shows the replication status of a file space on the secondary server:

```
filespace query
Filespace pattern to query:*
Are the above responses correct (y/n/q)?

Filespace Name  Type            Occupancy  Capacity   Start         End
--------------- --------------- ---------- ---------- ------------- -------------
/fs             API:Sample             100        300   0/0/0 0:0:0   0/0/0 0:0:0

     Start of last Replication :  2013/5/21 21:3:2
     End of last Replication   :  2013/5/21 21:3:3
                                  Server                    Local
     Last backup store date    :  2013/5/21 21:18:25        2013/5/21 21:18:25
     Last archive store date   :  0/0/0 0:0:0               0/0/0 0:0:0
     Last HSM store date       :  0/0/0 0:0:0               0/0/0 0:0:0
     FSINFO : Sample API FS Info
```

Related reference:

"dsmGetNextQObj" on page 107

Example flow diagrams for backup and archive

The API is designed for straightforward logic flows and clear transitions between the various states of the application client. This clean state transition catches logic flaws and program errors early in the development cycle, greatly enhancing the quality and reliability of the system.

For example, you cannot make a **dsmSendObj** call unless a transaction was started and a **dsmBindMC** call was previously made for the object that you are backing up.

Figure 12 on page 59 displays the state diagram for performing backup or archive operations within a transaction. The arrow pointing from "In Send Object" to **dsmEndTxn** indicates that a **dsmEndTxn** call can be started after a call to **dsmSendObj** or **dsmSendData**. You might want to do this if an error condition occurred during the send of an object and you want to stop the entire operation. In this case, you must use a vote of DSM_VOTE_ABORT. In normal circumstances, however, call **dsmEndSendObj** before you end the transaction.

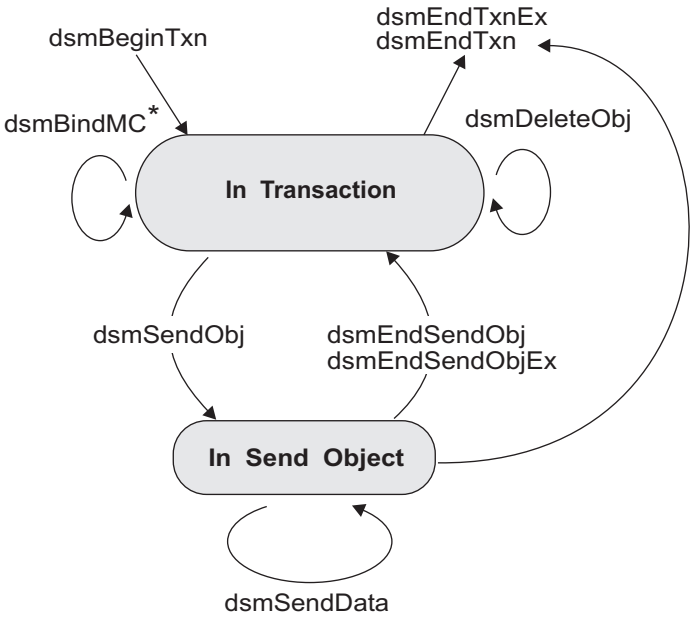


Figure 12. State diagram for backup and archive operations

Figure 13 on page 60 displays the flowchart for performing backup or archive operations within a transaction.

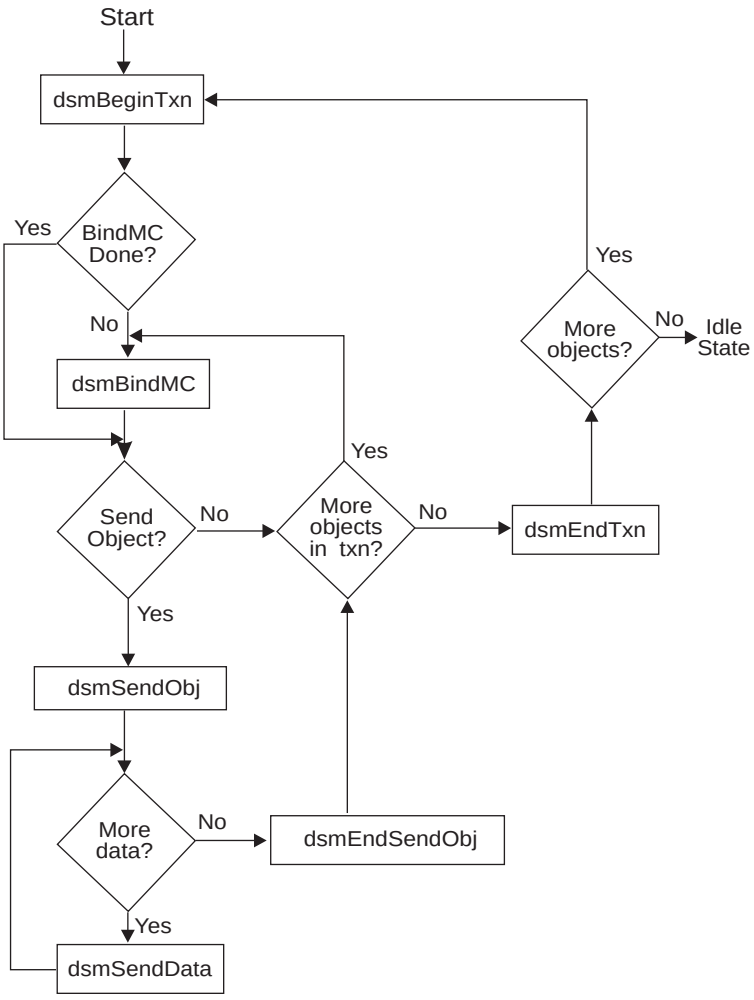


Figure 13. Flowchart for backup and archive operations

The primary feature in these two diagrams is the loop between the following API calls from within a transaction:

- **dsmBindMC**
- **dsmSendObj**
- **dsmSendData**
- **dsmEndSendObj**

The **dsmBindMC** call is unique in that you can start it from inside or outside of a transaction boundary. You can also start it from a different transaction, if required. The only requirement for the **dsmBindMC** call is that it is made prior to backing up or archiving an object. If the object that you are backing up or archiving is not associated with a management class, an error code is returned from **dsmSendObj**. In this situation, the transaction is ended by calling **dsmEndTxn** (this error condition is not shown in the flowchart).

The flowchart illustrates how an application would use multiple object transactions. It shows where decision points can be placed to determine if the object that is sent fits within the transaction or whether to start a new transaction.

Code example of API functions that send data to Tivoli Storage Manager storage

This example demonstrates the use of the API functions that send data to IBM Tivoli Storage Manager storage. The **dsmSendObj** call appears inside a switch statement, so that different parameters can be called depending on whether a backup or archive operation is being performed.

The **dsmSendData** call is called from inside a loop that repeatedly sends data until a flag is set that permits the program execution to exit the loop. The entire send operation is performed from within the transaction.

The third parameter on the **dsmSendObj** call is a buffer that contains the archive description. Because backup objects do not have a description, this parameter is NULL when backing up an object.

Figure 8 on page 30 displays an example that shows the use of the **dsmBindMC** function call.

```
if ((rc = dsmBeginTxn(dsmHandle)) )       /* API session handle  */
{
   printf("*** dsmBeginTxn failed: ");
   rcApiOut(dsmHandle, rc);
   return;
}

/*  Call dsmBindMC if not done previously */
objAttr.sizeEstimate.hi = 0;       /* estimate of        */
objAttr.sizeEstimate.lo = 32000;   /*        object size */
switch (send_type)
{
   case (Backup_Send) :
      rc = dsmSendObj(dsmHandle,stBackup,
      NULL,&objName,&objAttr,NULL);
      break;
   case (Archive_Send) :
      archData.stVersion = sndArchiveDataVersion;
      archData.descr = desc;
      rc = dsmSendObj(dsmHandle,stArchive,
   &archData,&objName,&objAttr,NULL);
      break;
   default : ;
}
if (rc)
{
   printf("*** dsmSendObj failed: ");
   rcApiOut(dsmHandle, rc);
   return;
}
done = bFalse;
while (!done)
{
   dataBlk.stVersion = DataBlkVersion;
   dataBlk.bufferLen = send_amt;
   dataBlk.numBytes  = 0;
   dataBlk.bufferPtr = bkup_buff;
   rc = dsmSendData(dsmHandle,&dataBlk);
   if (rc)
   {
      printf("*** dsmSendData failed: ");
      rcApiOut(dsmHandle, rc);
      done = bTrue;
   }
   /* Adjust the dataBlk buffer for the next piece to send */
}
rc = dsmEndSendObj(dsmHandle);
if (rc)
{
   printf("*** dsmEndSendObj failed: ");
   rcApiOut(dsmHandle, rc);
}
txn_reason = 0;
rc = dsmEndTxn(dsmHandle,           /* API session handle       */
            DSM_VOTE_COMMIT,      /* Commit transaction       */
            &txn_reason);         /* Reason if txn aborted    */
if (rc || txn_reason)
{
   printf("*** dsmEndTxn failed: rc = ");
   rcApiOut(dsmHandle, rc);
   printf("   reason =
}
```

Figure 14. An example of sending data to a server

File grouping

The IBM Tivoli Storage Manager API has a logical file grouping protocol that relates several individual objects together. You can reference and manage these groups as a logical group on the server. A logical group requires that all group members and the group leader belong to the same node and file space on the server.

Each logical group has a group leader. If the group leader is deleted, the group is deleted. You cannot delete a member that is part of a group. Expiration of all members in a group is dependent on the group leader. For example, if a member is marked for expiration, the member does not expire unless the group leader expires. However, if a member is not marked for expiration, and the group leader is expired, then all members are expired.

File groups contain backup data only, and cannot contain archive data. Archive objects can use the **Archive Description** field to facilitate a type of grouping if required by an application.

The **dsmGroupHandler** call groups the operations. The **dsmGroupHandler** function must be called from within a transaction. Most group error conditions are caught on either the **dsmEndTxn**l or **dsmEndTxnEx** calls.

The out structure in **dsmEndTxnEx** includes a new field, **groupLeaderObjId**. This field contains the object ID of the group leader if a group was opened in that transaction. You can create a group across more than one transaction. A group is not committed, or saved, on the server until a close is performed. The **dsmGroupHandler** is an interface that can accept five different operations. They include:

- DSM_GROUP_ACTION_OPEN
- DSM_GROUP_ACTION_CLOSE
- DSM_GROUP_ACTION_ADD
- DSM_GROUP_ACTION_ASSIGNTO
- DSM_GROUP_ACTION_REMOVE

Table 16 lists the **dsmGroupHandler** function call actions:

Table 16. dsmGroupHanlder functions

Action	Description
OPEN	The OPEN action creates a group. The next object that is sent becomes the group leader. The group leader cannot have content. All objects after the first object become members that are added to the group. To create a group, open a group and pass in a unique string to identify the group. This unique identifier allows several groups with the same name to be opened. After the group is opened, the next object that is sent is the group leader. All other objects that are sent are group members.

Table 16. dsmGroupHanlder functions (continued)

Action	Description
CLOSE	The CLOSE action commits and saves an open group. To close the group, pass in the object name and the unique string that is used in the open operation. The application must check for open groups and, if necessary, close or delete the groups. A group is not committed or saved until the group is closed. A CLOSE action fails in the following conditions: • The group that you are trying to close has the same name as an existing open group. • A management class incompatibility exists between the current closed group and the new group to be closed of the same name. In this case, complete the following steps: 1. Query the previous closed group. 2. If the management class of the existing closed group is different from the management class associated with the current open group, issue a **dsmUpdateObject** with type DSM_BACKUPD_MC. This command updates the existing group to the new management class. 3. Issue the CLOSE action.
ADD	The ADD action appends an object to a group. All objects that are sent after the ADD action are assigned to the group.
ASSIGNTO	The ASSIGNTO action permits the client to assign objects that exist on the server to the declared peer group. This transaction sets up the PEER group relationship. The ASSIGNTO action is similar to the ADD action, with the following exceptions: • The ADD action applies to objects within an in-flight transaction. • The ASSIGNTO action applies to an object that is on the server.
REMOVE	The REMOVE action removes a member, or a list of members, from a group. A group leader cannot be removed from a group. A group member must be removed before the member can be deleted.

Use the following query types for group support:
- **qtBackupGroups**
- **qtOpenGroups**

The **qtBackupGroups** queries groups that are closed while **qtOpenGroups** queries groups that are open. The query buffer for the new types has fields for **groupLeaderObjId** and **objType**. The query performs differently depending on the values for these two fields. The following table includes some query possibilities:

Table 17. Examples of queries

groupLeaderObjId.hi	groupLeaderObjId.lo	objType	Result
0	0	NULL	Returns a list of all group leaders
grpLdrObjId.hi	grpLdrObjId.lo	0	Returns a list for all group members that are assigned to the specified group leader (**grpLdrObjId**).
grpLdrObjId.hi	grpLdrObjId.lo	objType	Returns a list by using **BackQryRespEnhanced3**, for each group member that is assigned to the specified group leader (**grpLdrObjId**), and matching the object type (**objType**).

The response structure (**qryRespBackupData**) from **dsmGetNextQObj** includes two fields for group support:

- **isGroupLeader**
- **isOpenGroup**

These fields are Boolean flags. The following example displays the creation of the group, adding members to the group, and closing the group to commit the group on the Tivoli Storage Manager server.

```
dsmBeginTxn
 dsmGroupHandler (PEER, OPEN, leader, uniqueId)
 dsmBeginSendObj
  dsmEndSendObj
dsmEndTxnEx (With objId of leader)
Loop for multiple txns
{
 dsmBeginTxn
  dsmGroupHandler (PEER, ADD, member, groupLeaderObjID)
  Loop for multiple objects
  {
   dsmBeginSendObj
   Loop for data
   {
    dsmSendData
   }
   dsmEndSendObj
  }
 dsmEndTxn
}
dmBeginTxn
 dsmGroupHandler(CLOSE)
dsmEndTxn
```

Figure 15. Example of pseudo-code that is used to create a group

For a code example, see the sample group program `dsmgrp.c` that is included in the API `sampsrc` directory.

Receiving data from a server

Application clients can receive data or named objects and their associated data from IBM Tivoli Storage Manager storage by using the restore and retrieve functions. The restore function accesses objects that previously were backed up, and the retrieve function accesses objects that previously were archived.

Restriction: The API can only restore or retrieve objects that were backed up or archived using API calls.

Both restore and retrieve functions start with a query operation. The query returns different information depending on whether the data was originally backed up or archived. For instance, a query on backup objects returns information on whether an object is active or inactive, while a query on archive objects returns information such as object descriptions. Both queries return object IDs that are used to uniquely identify the object on the server.

Partial object restore or retrieve

The application client can receive only a portion of the object. This is called a partial object restore or a partial object retrieve.

Attention: Partial restore or retrieve of compressed or encrypted objects produces unpredictable results.

Note: If you code your application to use a partial object restore or retrieve, you cannot compress the data while sending it. To enforce this, set *ObjAttr.objCompressed* to *bTrue*.

To perform a partial object restore or retrieve, associate the following two data fields with each object **GetList** entry:

offset The byte offset into the object from which to begin returning data.

length The number of object bytes to return.

Use DSM_MAX_PARTIAL_GET_OBJ to determine the maximum number of objects that can perform a partial object restore or retrieve for a specific **dsmBeginGetData** list.

The following data fields, used on the **dsmBeginGetData** call, determine what portion of the object is restored or retrieved:

- If both the offset and length are zero, the entire object is restored or retrieved from Tivoli Storage Manager storage.
- If the offset is greater than zero, but the length is zero, the object is restored or retrieved from the offset to the end.
- If the length is greater than zero, only the portion of the object from the offset for the specified length is restored or retrieved.

Restoring or retrieving data

After a query is made and a session is established with the IBM Tivoli Storage Manager server, you can run a procedure to restore or retrieve data.

To restore or retrieve data, complete the following steps:

1. Query the Tivoli Storage Manager server for either backup or archive data.
2. Determine the objects to restore or retrieve from the server.
3. Sort the objects on the **Restore Order** field.
4. Send the **dsmBeginGetData** call with the list of objects that you want to access.
5. Send the **dsmGetObj** call to obtain each object from the system. Multiple **dsmGetData** calls might be needed for each object to obtain all associated object data. Send the **dsmEndGetObj** call after all data for an object is obtained.
6. Send the **dsmEndGetData** call after all data for all objects is received, or to end the receive operation.

Querying the server

Before you begin any restore or retrieve operation, first query the IBM Tivoli Storage Manager server to determine what objects you can receive from storage.

To send the query, the application must enter the parameter lists and structures for the **dsmBeginQuery** call. The structure must include the file space that the query examines and pattern-match entries for the high-level and low-level name fields. If the session was initialized with a NULL owner name, you do not need to specify the owner field. However, if the session was initialized with an explicit owner name, only objects that are associated with that owner name are returned.

The point-in-time **BackupQuery** query provides a snapshot of the system at a specific time. By specifying a valid date, you can query all files that are backed up to that time. Even if an object has an active backup from a later date, point-in-time overrides an object state so that the previous inactive copy is returned. For more information, see the following example: pitDate. You must be connected to at least a Version 3 server to use the point-in-time **BackupQuery** query.

A query returns all information that is stored with the object, in addition to the information in the following table.

Table 18. Query to the server return information

Field	Description
`copyId`	The `copyIdHi` and `copyIdLo` values provide an 8-byte number that uniquely identifies this object for this node in Tivoli Storage Manager storage. Use this ID to request a specific object from storage for restore or retrieve processing.
`restoreOrderExt`	The `restoreOrderExt` value provides a mechanism for receiving objects from Tivoli Storage Manager storage in the most efficient manner possible. Sort the objects to restore on this value to ensure that tapes are mounted only one time and are read from front to back.

You must keep some or all of the query information for later processing. Keep the `copyId` and `restoreOrderExt` fields because they are needed for the actual restore operation. You must also keep any other information needed to open a data file or identify a destination.

Call **dsmEndQuery** to finish the query operation.

Selecting and sorting objects by restore order

After the backup or archive query is performed, the application client must determine which objects, if any, are to be restored or retrieved.

Then you sort the objects in ascending order (low to high). This sorting is very important to the performance of the restore operation. Sorting the objects on the **restoreOrderExt** fields ensures that the data is read from the server in the most efficient order.

All data on disk is restored first, followed by data on media classes that require volume mounts (such as tape). The **restoreOrderExt** field also ensures that data on tape is read in order with processing starting at the front of a tape and progressing towards the end.

Properly sorting on the **restoreOrderExt** field means that duplicate tape mounts and unnecessary tape rewinds do not occur.

A non-zero value in the **restoreOrderExt**.top field correlates to a unique serial access device on the IBM Tivoli Storage Manager server. Since a serial access device can only be used by one session / mount point at a time, the application should ensure that if it uses multiple sessions there are not concurrent restores with the same **restoreOrderExt**.top value. Otherwise the first session are able to access the objects, but other sessions wait until the first session terminates and the device becomes available.

The following example shows how to sort objects by using **Restore Order** fields.

Figure 16. Sorting objects with the restore order fields

```
typedef struct {
dsStruct64_t      objId;
dsUint160_t      restoreOrderExt;

}  SortOrder;               /* struct used for sorting */

=====================================================================
/* the code for sorting starts from here */
dsmQueryType      queryType;
qryBackupData     queryBuffer;
DataBlk           qDataBlkArea;
qryRespBackupData qbDataArea;
dsInt16_t   rc;
dsBool_t  done = bFalse;
int i = 0;
int qry_item;
SortOrder  sortorder[100]; /* sorting can be done up to 100 items
                              only right now.  Set appropriate
                              array size to fit your needs */
/*-------------------------------------------------------------------+
| NOTE: Make sure that proper initializations have been done to
|  queryType,
|       queryBuffer, qDataBlkAre, and qbDataArea.
|
-------------------------------------------------------------------*/

   qDataBlkArea.bufferPtf = (char*) &qbDataArea;

   rc = dsmBeginQuery(dsmHandle, queryType, (void *) &queryBuffer);

   /*---------------------------------------+
   | Make sure to check rc from dsmBeginQuery
   +---------------------------------------*/
   while (!done)
   {
      rc = dsmGetNextQObj(dsmHandle, &qDataBlkArea);
 if ((rc == DSM_RC_MORE_DATA) ||
          (rc == DSM_RC_FINISHED))
           &&( qDataBlkArea.numBytes))
      {
         /*****************************************/
         /* transferring restoreOrderExt and objId */
         /*****************************************/
         sortorder[i].restoreOrderExt = qbDataArea.restoreOrderExt;
         sortorder[i].objId = qbDataArea.objId;

      } /*  if ((rc == DSM_RC_MORE_DATA) || (rc == DSM_RC_FINISHED)) */
      else
      {
```

```
            done = bTrue;
            /****************************/
            /* take appropriate action. */
            /****************************/
         }

         i++;
         qry_item++;

      } /* while (!done) */
      rc = dsmEndQuery(dsmHandle);
     /*check rc  */
      /*****************************************************/
      /* sorting the array using qsort.  After the call,   */
      /* sortorder will be sorted by restoreOrderExt field */
      /*****************************************************/

      qsort(sortorder, qry_item, sizeof(SortOrder), SortRestoreOrder);

 /*-------------------------------------------------------------------+
 | NOTE: Make sure to extract sorted object ids and store them in
 |       any data structure you want.
 -------------------------------------------------------------------*/

 /*-------------------------------------------------------------------+
 | int SortRestoreOrder(SortOrder *a, SortOrder *b)
 |
 | This function compares restoreOrder fields from two structures.
 | if (a > b)
 |   return(GREATERTHAN);
 | if (a < b)
 |   return(LESSTHAN);
 | if (a == b)
 |   return(EQUAL);
 +-------------------------------------------------------------------*/
 int SortRestoreOrder(SortOrder *a, SortOrder *b)
 {
     if (a->restoreOrderExt.top > b->restoreOrderExt.top)
        return(GREATERTHAN);
     else if (a->restoreOrderExt.top < b->restoreOrderExt.top)
        return(LESSTHAN);
     else if (a->restoreOrderExt.hi_hi > b->restoreOrderExt.hi_hi)
        return(GREATERTHAN);
     else if (a->restoreOrderExt.hi_hi < b->restoreOrderExt.hi_hi)
        return(LESSTHAN);
     else if (a->restoreOrderExt.hi_lo > b->restoreOrderExt.hi_lo)
        return(GREATERTHAN);
     else if (a->restoreOrderExt.hi_lo < b->restoreOrderExt.hi_lo)
        return(LESSTHAN);
      else if (a->restoreOrderExt.lo_hi > b->restoreOrderExt.lo_hi)
        return(GREATERTHAN);
     else if (a->restoreOrderExt.lo_hi < b->restoreOrderExt.lo_hi)
        return(LESSTHAN);
     else if (a->restoreOrderExt.lo_lo > b->restoreOrderExt.lo_lo)
        return(GREATERTHAN);
     else if (a->restoreOrderExt.lo_lo < b->restoreOrderExt.lo_lo)
        return(LESSTHAN);
     else
        return(EQUAL);
 }
```

Starting the dsmBeginGetData call

After you select and sort the objects to receive, submit them to IBM Tivoli Storage Manager for either a restore or retrieve operation. The **dsmBeginGetData** call begins a restore or retrieve operation. The objects are returned to the application client in the order you requested.

Complete the information for these two parameters in these calls:

mountWait
This parameter tells the server whether the application client waits for offline media to be mounted in order to obtain data for an object, or whether that object should be skipped during processing of the restore or retrieve operation.

dsmGetObjListP
This parameter is a data structure that contains the **objId** field which is a list of all object IDs that are restored or retrieved. Each **objId** is associated with a **partialObjData** structure that describes whether the entire **objId** or only a particular section of the object will be retrieved.

Each **objId** is eight bytes in length, so a single restore or retrieve request can contain thousands of objects. The number of objects to request in a single call is limited to DSM_MAX_GET_OBJ or DSM_MAX_PARTIAL_GET_OBJ.

Receiving each object to restore or retrieve

After the **dsmBeginGetData** call is sent, you can perform a procedure to receive each object that is sent from the server.

The DSM_RC_MORE_DATA return code means that a buffer was returned and that you should call **dsmGetData** again. Check the **DataBlk.numBytes** for the actual number of returned bytes.

When you obtain all data for an object, you must send a **dsmEndGetObj** call. If more objects will be received, send **dsmGetObj** again.

If you want to stop the process, for example, to discard any remaining data in the restore stream for all objects that are not yet received, send the **dsmEndGetData** call. This call flushes the data from the server to the client. However, using this method might take time to complete. If you want to end a restore operation, use **dsmTerminate** to close the session.

1. Send the **dsmGetObj** call to identify the object that you requested from the data stream and to obtain the first block of data that is associated with the object.
2. Send more **dsmGetData** calls, as necessary to obtain the remaining object data.

Example flow diagrams for restore and retrieve

A state diagram and a flowchart can be used to illustrate how to perform restore or retrieve operations.

The arrow pointing from "In Get Object" to **dsmEndGetData** indicates that you can send a **dsmEndGetData** call after a call to **dsmGetObj** or **dsmGetData**. You might need to do this if an error condition occurred while getting an object from IBM Tivoli Storage Manager storage and you want to stop the operation. In normal circumstances, however, call **dsmEndGetObj** first.

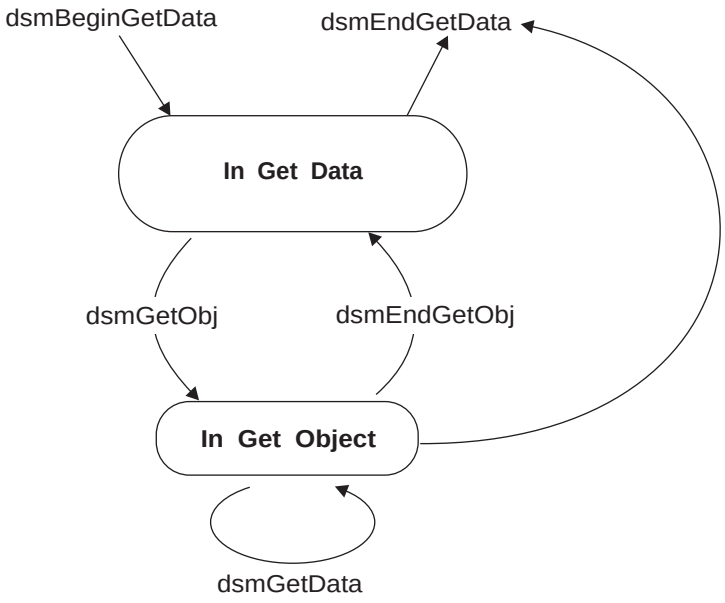


Figure 17. State diagram for restore and retrieve operations

Figure 18 displays the flowchart for performing restore or retrieve operations.

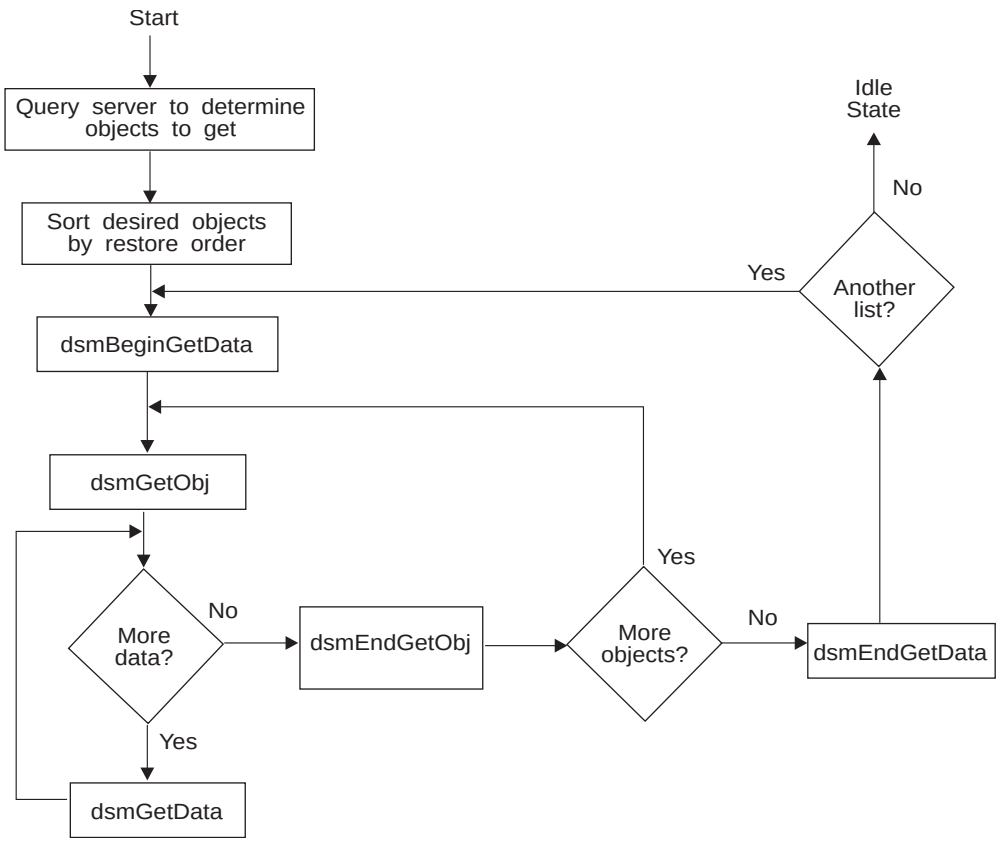


Figure 18. Flowchart for restore and retrieve operations

Code example of receiving data from a server

This example demonstrates using the API functions to retrieve data from Tivoli Storage Manager storage.

The **dsmBeginGetData** function call appears inside a switch statement, so that different parameters can be called depending on whether a restore or retrieve operation is being performed. The **dsmGetData** function call is called from inside a loop that repeatedly gets data from the server until a flag is set that permits the program execution to exit the loop.

Figure 19. An example of receiving data from a server

```
/* Call dsmBeginQuery and create a linked list of objects to restore. */
/* Process this list to create the proper list for the GetData calls. */
/* Set up the getList structure to point to this list.                */
/* This example is set up to perform a partial object retrieve.  To   */
/* retrieve only complete objects, set up:                            */
/*       getList.stVersion = dsmGetListVersion;                       */
/*       getList.partialObjData = NULL;                               */
dsmGetList getList;
getList.stVersion = dsmGetListPORVersion;   /* structure version      */
getList.numObjId  = items;               /* number of items in list   */
getList.objId     = (ObjID *)rest_ibuff;
                                    /* list of object IDs to restore  */
getList.partialObjData = (PartialObjData *) part_ibuff;
                                     /* list of partial object data  */
switch(get_type)
{
   case (Restore_Get) :
     rc = dsmBeginGetData(dsmHandle,bFalse,gtBackup,&getList);
     break;
   case (Retrieve_Get) :
     rc = dsmBeginGetData(dsmHandle,bFalse,gtArchive,&getList);
     break;
   default : ;
}
if (rc)
{
   printf("*** dsmBeginGetData failed: ");
   rcApiOut(dsmHandle, rc);
   return rc;
}
/* Get each object from the list and verify whether it is on the  */
/* server.  If so, initialize structures with object attributes for */
/* data validation checks.  When done, call dsmGetObj.             */
 rc = dsmGetObj(dsmHandle,objId,&dataBlk);
done = bFalse;
while(!done)
{
   if (   (rc == DSM_RC_MORE_DATA)
       || (rc == DSM_RC_FINISHED))
   {
      if (rc == DSM_RC_MORE_DATA)
      {
         dataBlk.numBytes = 0;
         rc = dsmGetData(dsmHandle,&dataBlk);
      }
      else
         done = bTrue;
   }
   else
   {
```

```
      printf("*** dsmGetObj or dsmGetData failed: ");
      rcApiOut(dsmHandle, rc);
      done = bTrue;
   }
} /* while */
rc = dsmEndGetObj(dsmHandle);
/* check rc from dsmEndGetObj */
/* check rc from dsmEndGetData */
rc = dsmEndGetData(dsmHandle);
return 0;
```

Updating and deleting objects on the server

Your API applications can use the**dsmUpdateObj** or **dsmUpdateObjEx** function call to update objects that were archived or backed up. Use either call in the session state only, updating one object at a time. Use **dsmUpdateObjEx** to update any of several archive objects containing the same name.

To select an archive object, set the **dsmSendType** function call to **stArchive**.

- With **dsmUpdateObj**, only the latest archive object with the assigned name is updated.
- With **dsmUpdateObjEx**, any archived object can be updated by specifying the proper object ID.

For an archived object, the application can update the following fields:

- Description
- Object information
- Owner

To select a backup object, set **dsmSendType** to **stBackup**. For backed-up objects, only the active copy is updated.

For a backed-up object, the application can update the following fields:

- Management class
- Object information
- Owner

Deleting objects from the server

API applications can make calls to either delete objects that were archived or turn off objects that were backed up. Deleting archived objects is dependent on the node authorization that was given when the administrator registered the node. Administrators can specify that nodes can delete archived objects.

Use the **dsmDeleteObj** function call to delete archived objects and turn off backup objects. Using this **delType** removes the backup object from the server. This is based on **objID**, deletes an object from the server database. Only an owner of an object can delete it. You can delete any version (active or inactive) of an object. The server reconciles the versions. If you delete an active version of an object, the first inactive version becomes active. If you delete an inactive version of an object, all older versions advance. The node must be registered with **backDel** permission.

An archived object is marked for deletion in storage when the system performs its next object expiration cycle. Once you delete an archived object from the server, you cannot retrieve it.

When you inactivate a backup object at the server, the object moves from an active state to an inactive state. These states have different retention policies associated with them that are based on the management class that is assigned.

Similar to the **dsmSendObj** call, a call to **dsmDeleteObj** is sent within the boundary of a transaction. The state diagram in Figure 12 on page 59 displays how a call to **dsmDeleteObj** is preceded by a call to **dsmBeginTxn** and followed by a call to **dsmEndTxn**.

Logging events

An API application can log event messages to central locations. The application can direct logging to the IBM Tivoli Storage Manager server, the local machine, or both. The **dsmLogEventEx** function call is performed in a session. To view messages logged on the server, use the query **actlog** command through the administrative client.

Use the Tivoli Storage Manager client option, `errorlogretention`, to prune the client error log file if the application writes numerous client messages to the client log `dsmLogType`, either `logLocal` or `logBoth`.

For more information about Tivoli Storage Manager logs, see the Tivoli Storage Manager server documentation.

State diagram summary for the Tivoli Storage Manager API

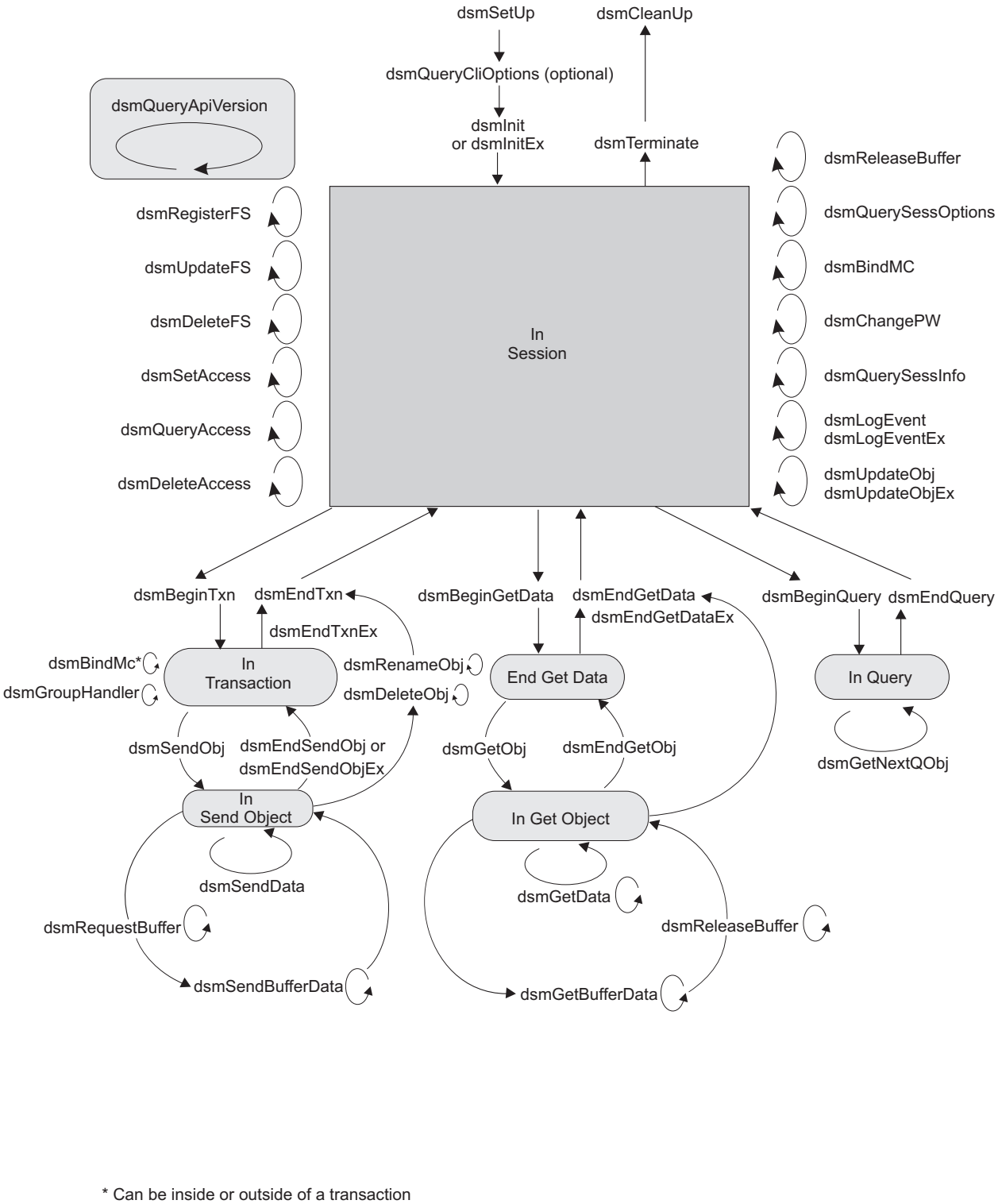
Once you review all the considerations for creating your own application with the Tivoli Storage Manager API, review this state diagram summary of an entire application.

Figure 20 on page 75 contains the state diagram for the API. It contains all previously displayed state diagrams in addition to several other calls previously not displayed.

The points in this diagram include:

- Call **dsmQueryApiVersionEx** at any time. It has no state associated with it. See Figure 1 on page 15 for an example.
- Call **dsmQueryCliOptions** before a **dsmInitEx** call only.
- Use **dsmRegisterFS**, **dsmUpdateFS**, and **dsmDeleteFS** to manage file spaces. These calls are made from within an idle session state. Use the **dsmBeginQuery** call to query file spaces. For more information about file space calls, see "Managing file spaces" on page 26.
- Send the **dsmBindMC** call from within an idle session state or from within a send object transaction state. See the example in Figure 8 on page 30.
- Send the **dsmChangePW** call from within an idle session state.

  Note: If the **dsmInitEx** call returns with a password-expired return code, the **dsmChangePW** call must be made before you start a valid session. See Figure 4 on page 21 for an example that uses **dsmChangePW**.
- If a call returns with an error, the state remains as it was. For example, if **dsmGetObj** returns with an error, the state remains In Get Data, and a call to **dsmEndGetObj** is a call sequence error.



* Can be inside or outside of a transaction

Figure 20. Summary state diagram for the API

Chapter 4. Understanding interoperability

The API has two types of interoperability: between the backup-archive client and API applications and between different operating systems.

Backup-archive client interoperability

The backup-archive command line can access API objects to provide limited interoperability. API objects can only be viewed and accessed from the backup-archive command line client and cannot be viewed or accessed from any of the graphical interfaces. The backup-archive command-line client can only restore content of the file and nothing else, so you should only use it for a salvage type of operation.

The following command-line actions are provided:
- Delete archive
- Delete filespace
- Query
- Restore
- Retrieve
- Set access

The path information is actual directories for backup-archive client objects. In contrast, the API object path information might not have any relationship to existing directories: the path might be completely contrived. Interoperability does not change this aspect of these object types. To use this feature successfully, follow the restrictions and conventions.

Notes:

1. There is no interoperability between the backup-archive client and API objects stored on a retention protection server.
2. You cannot use the backup-archive client GUIs to access files that were stored using the API client. You can only use the command line to access these files.

Naming your API objects

Establish a consistent naming convention for API object names. The naming convention must cater for the file space name, the high-level qualifier, and the low-level qualifier. The file space name and high-level qualifiers can refer to actual directory names. Each object name can consist of more than one directory name that applies to the low-level qualifier.

For convenience, use the name of the object that is not prefixed with directory information as the low-level qualifier. For more information, see “Object names and IDs” on page 23.

File space names must be fully qualified when they are referred to from either the API or the backup-archive command line. For example, on a UNIX or Linux operating system, you register the following file spaces:
- `/a`
- `/a/b`

© Copyright IBM Corp. 1993, 2016

When you refer to /a, objects that are related only to file space /a are displayed. To view objects that are related to /a/b, you must specify /a/b as the file space name.

After you register both file spaces, if you back up object b into file space /a, then a query for /a/b continues to display objects that are related only to file space /a/b.

The exception to this restriction occurs in file space references when you attempt to query or delete file spaces with the API. In both cases, the file space names do not have to be fully qualified if you use a wildcard character. For example, /a* refers to both /a and /a/b.

Tip: If interoperability is important for you, then avoid file space names that overlap.

On Windows systems, enclose file space names in braces { } for API objects when you access the objects from the backup-archive command line interface. Windows operating systems automatically place file space names in uppercase letters when you register or refer the names. However, this automatic function does not occur for the remainder of the object name specification. If you want full interoperability, place the high-level qualifier and the low-level qualifier in uppercase letters in the application when you back up API objects. If your application does not uppercase high-level qualifiers (directory names) and low-level qualifiers (file names) before it sends objects to the server, you will be unable to access the objects directly by name through the backup-archive client.

For example, if an object is stored on the server as {"FileSpacename"}\TEST\MYDIRNAME\file.txt, you cannot directly restore or query the file.txt object because your application did not uppercase the file name before the file was copied to the server. The only way to manipulate these objects is to use wildcard characters. For example, to query \TEST\MYDIRNAME\file.txt, a backup-archive client user must use wildcard characters for all parts of the object name that were not uppercased before they were sent to the server. The following command must be used to query this file.txt file:

```
dsmc query backup {"FileSpaceName"}\TEST\MYDIRNAME\*
```

If any other of the other qualifiers are also saved in lowercase text, those qualifiers must also be queried by using wildcards. For example, to query an object that is stored as {"FileSpaceName"}\TEST\mydirname\file.txt, use the following command:

```
dsmc query backup {"FileSpaceName"}\TEST\*\*
```

The examples that follow demonstrate these concepts. In both Windows and UNIX or Linux environments, you do not have to specify either the complete high-level or low-level qualifier. However, if you do not specify the complete qualifier, then you must use the wildcard character.

Platform	Example
Windows	To query all backed-up files in file space MYFS, enter the following string: `dsmc q ba "{MYFS}\*\*"` You must use at least one asterisk (*) for each of the high-level and low-level qualifiers.

Platform	Example
UNIX or Linux	To query all backed-up files in file space /A, enter the following string: `dsmc q ba "/A/*/*"` You must use at least one asterisk (*) for each of the high-level and low-level qualifiers.

Backup-archive client commands you can use with the API

You can use a subset of backup-archive client commands within an application. For example, you can view and manage objects that other users own either on the same node or on a different node.

To view and manage objects that other users own either on the same node or on a different node, perform these steps:

1. Give access with the **set access** command.
2. Specify the owner and the node. Use the *fromowner* and *fromnode* options from the backup-archive command line to specify the owner and the node. For example:

```
dsmc q ba "/A/*/*" -fromowner=other_owner -fromnode=other_node
```

Table 19 describes the commands that you can use with API objects.

Table 19. Backup-archive client commands you can use with API objects

Command	Description
Delete Archive	Archived files that the current user owns can be deleted. The set access command settings have no effect on this command.
Delete Filespace	The **delete filespace** command affects API objects.
Query	From the backup-archive command line, you can query backed up and archived API objects and objects that other users own, or that exist on other nodes. See "Naming your API objects" on page 77 for information about querying API objects. Use the existing *–fromowner* option to query objects that a different user owns for which the set access permission has been given. Use the existing *–fromnode* option to query objects that exist on another node for which the set access permission has been given. For more information, see "**dsmInitEx**" on page 115.
Restore **Retrieve**	**Note:** Use these commands only for exception situations. API objects that are encrypted using the application managed key can be restored or retrieved if the encryption key is known or saved in the password file or registry. API objects encrypted by using transparent encryption cannot be restored or retrieved by using the backup-archive client. These commands return data as bit files that are created by using default file attributes. You can restore or retrieve API objects that other users own, or that are from a different node. The set access command determines which objects qualify.
Set Access	The **set access** command permits users to manage API objects that another user owns, or that are from another node.

Operating system interoperability

The IBM Tivoli Storage Manager API supports cross-platform interoperability. Applications on a UNIX or Linux system can operate on file spaces and objects that are backed up from a Windows system. Similarly, a Windows system can operate on file spaces and objects that are backed up from a UNIX or Linux system.

By default, the names of objects from one UNIX system are compatible with the names of objects from other UNIX systems. By default, names of objects from Windows systems are not compatible with names of objects from UNIX systems. Several parameters control the naming of objects in Tivoli Storage Manager file spaces. If you set up an application appropriately, the names of objects can be used by applications that run on both Windows systems and UNIX systems. Use the same parameters to back up and restore objects.

Restriction: A Windows application that uses Unicode creates a file space that is not compatible with applications that run on UNIX systems.

To achieve interoperability, complete the following setup tasks:

1. Establish a consistent naming convention. Select a character for the `dir` delimiter, such as forward slash (`/`) or backslash (`\`). Place the directory delimiter character in front of the file space name, the high-level qualifier, and the low-level qualifier.
2. When you call **`dsmInitEx`**, set the value of the **`dirDelimiter`** field to the directory delimiter character that you selected and set **`bCrossPlatform`** to **`bTrue`**.
3. Set the **`useUnicode`** flag to **`bFalse`** when you use the Tivoli Storage Manager interface. Unicode file names are not compatible with non-Unicode file names.

Backing up multiple nodes with client node proxy support

Backups of multiple nodes which share storage can be consolidated to a common target node name on the IBM Tivoli Storage Manager server. This method is useful when the system that runs the backup can change over time, such as with a cluster. You can also use the `asnodename` option to restore data from a different system other than the one which ran the backup.

Use the `asnodename` option on the **`dsmInitEx`** option string to back up, archive, restore, and retrieve, query, or delete data under the target node name on the Tivoli Storage Manager server. You can also specify the `asnodename` option in the `dsm.opt` or `dsm.sys` file.

Restriction: Do not use target nodes as traditional nodes, especially if you encrypt your files before you back up to the server.

To enable this option, complete the following steps:

1. Install the API client on all nodes in a shared data environment.
2. If not already registered, register each node with the Tivoli Storage Manager server. Register the common "target" node name to be shared by each of the agent nodes that are used in your shared data environment.
3. Register each of the agent nodes in the shared data environment with the server. The agent node name is used for authentication. Data is not stored by using the agent node name when the `asnodename` option is used.

4. Ask your administrator to grant proxy authority to all nodes in the shared environment to access the target node name on the Tivoli Storage Manager server, by using the **grant proxynode** command.
5. Use the **query proxynode** administrative client command to display the client nodes that have the authority to perform client operations on behalf of another node. This authority is granted by the **grant proxynode** command. Or use the **dsmQuery** command with the query type **qtProxyNodeAuth** to see the nodes to which this node can proxy.
6. If the application is using user encryption of data, not `TSMENCRKEY`, ensure that all nodes use the same encryption key. You must use the same encryption key for all files that are backed up in the shared node environment.

Related tasks:

Backing up data with client-node proxy support (UNIX and Linux systems)

Backing up data with client-node proxy support (Windows systems)

Chapter 5. Using the API with Unicode

The Tivoli Storage Manager API supports Unicode UCS2, a fixed length, double-byte code page that has code points for all known code pages, such as Japanese, Chinese, or German. It supports as many as 65,535 unique code points.

Restriction: This feature is only available on Windows.

With Unicode, your application can back up and restore file names in any character set from the same machine. For example, on an English machine, you can back up and restore file names in any other language code page.

When to use Unicode

You can simplify your application that supports multiple languages by writing a Unicode application and by taking advantage of the IBM Tivoli Storage Manager Unicode interface.

Use the Tivoli Storage Manager Unicode interface if any of the following conditions are true:

- If your application is already compiled for Unicode and it was converting to a multibyte character set (mbcs) before calling the Tivoli Storage Manager API.
- If you are writing a new application and want to enable your application to support Unicode.
- If your application uses a string passed to it from an operating system or other application that uses Unicode.

If you do not need Unicode, it is not necessary to compile your application again.

The API continues to support the dsm interface. The API SDK contains callmtu1.c and callmtu2.c sample programs that demonstrate how to use the Unicode API. Use **makemtu** to compile these programs.

Setting up Unicode

To set up and use Unicode you must perform a particular procedure so the API registers a Unicode file space on the server and all file names in that file space become Unicode strings.

Restriction: You cannot store Unicode and non-Unicode file names in the same file space.

1. Compile the code with the `-DUNICODE` flag.
2. All strings in your application must be wchar strings.
3. Follow the structures in the tsmapitd.h file, and the function definitions in the tsmapifp.h file for calls to the API.
4. Set the *useUnicode* flag to *bTrue* on the **tsmInitEx** function call. Any new file space is registered as a Unicode file space.

When you send data to previously registered, non-Unicode file spaces, the API continues to send file names as non-Unicode. Rename the old file spaces on the server to fsname_old and start a new Unicode file space for new data. The API

© Copyright IBM Corp. 1993, 2016

restores non-Unicode data from the old file spaces. Use the **bIsUnicode** field in the **tsmQryRespFSData** structure that is returned on a query file space to determine whether or not a file space is Unicode.

Each **dsmXXX** function call has a matching **tsmXXX** function call. The difference between the two are the structures that are used. All **tsmXXX** function call structures have dsChar_t types for string values when they are compiled with the UNICODE flag. The dsChar_r maps to wchar. There is no other difference between these interfaces.

Restriction: Use either one interface or the other. Do not mix the **dsmXXX** function call and **tsmXXX** function call interfaces. Ensure that you use the Tivoli Storage Manager structures and Tivoli Storage Manager version definitions.

Some constants continue to be defined in the dsmapitd.h file, so you need both the dsmapitd.h and the tsmapitd.h files when you compile.

You can use the Tivoli Storage Manager interface on other operating systems, such as UNIX or Linux, but on these operating systems, the dsChar_t type maps to char because Unicode is supported on only Windows operating systems. You can write only one variation of the application and compile on more than one operating system using the Tivoli Storage Manager interface. If you are writing a new application, use the Tivoli Storage Manager interface.

If you are upgrading an existing application:
1. Convert the **dsmXXX** function call structures and calls to the Tivoli Storage Manager interface.
2. Migrate existing file spaces.
3. Back up new file spaces with the *useUnicode* flag set to *true*.

Note: After you use a Unicode-enabled client to access a node, you cannot connect to the same node name with an older version of the API or with an API from another operating system. If your application uses cross-platform capability, do not use the `Unicode` flag. There is no cross-platform support between Unicode and non-Unicode operating systems.

When you enable the *useUnicode* flag, all string structures are treated as Unicode strings. On the server, only the following fields are true Unicode:
- File space name
- High level
- Low level
- Archive description

All remaining fields convert to mbcs in the local code page before they are sent to the server. Fields, such as nodename, are wchar strings. They must be valid in the current locale. For example, on a Japanese machine, you can back up files with Chinese names, but the node name must be a valid string in Japanese. The option file remains in the current code page. If you need to create a Unicode include-exclude list, use the *inclexcl* option with a file name and create a Unicode file with Unicode patterns in it.

Related reference:

inclexcl option

Chapter 6. API function calls

Table 20 provides an alphabetical list of the API function calls, a brief description and the location of more detailed information about the function call, which includes:

Element	Description
Purpose	Describes the function call.
Syntax	Contains the actual C code for the function call. This code is copied from the UNIX or Linux version of the dsmapifp.h header file. See Appendix C, "API function definitions source file," on page 197. This file differs slightly on other operating systems. Application programmers for other operating systems should check their version of the header file, dsmapifp.h, for the exact syntax of the API definitions.
Parameters	Describes each parameter in the function call, identifying it as either input (I) or output (O), depending on how it is used. Some parameters are designated as both input and output (I/O). The data types that are referenced in this section are defined in the dsmapitd.h header file. See Appendix B, "API type definitions source files," on page 155.
Return codes	Contains a list of the return codes that are specific to the function call. General system errors, such as communication errors, server problems, or user errors that might appear on any call are not listed. The return codes are defined in the dsmrc.h header file. See Appendix A, "API return codes source file: `dsmrc.h`," on page 145.

Table 20. API function calls

Function call and location	Description
"dsmBeginGetData" on page 87	Starts a restore or retrieve operation on a list of objects in storage.
"dsmBeginQuery" on page 89	Starts a query request to Tivoli Storage Manager for information.
"dsmBeginTxn" on page 93	Starts one or more transactions that begins a complete action. Either all of the actions succeed, or none succeed.
"dsmBindMC" on page 94	Associates, or binds, a management class to the object that is passed.
"dsmChangePW" on page 95	Changes a Storage Manager password.
"dsmCleanUp" on page 96	This call is used if **`dsmSetUp`** was called.
"dsmDeleteAccess" on page 97	Deletes current authorization rules for backup versions or archived copies of your objects.
"dsmDeleteFS" on page 97	Deletes a file space from storage.
"dsmDeleteObj" on page 98	Turns off backup objects, or deletes archive objects in storage.
"dsmEndGetData" on page 99	Ends a **`dsmBeginGetData`** session that gets objects from storage.
"dsmEndGetDataEx" on page 100	Provides the total of LAN-free bytes that were sent.
"dsmEndGetObj" on page 100	Ends a **`dsmGetObj`** session that obtains data for a specified object.

© Copyright IBM Corp. 1993, 2016

Table 20. API function calls (continued)

Function call and location	Description
"dsmEndQuery" on page 101	Signifies the end of a **dsmBeginQuery** action.
"dsmEndSendObj" on page 101	Indicates the end of data that is sent to storage.
"dsmEndSendObjEx" on page 102	Provides compression information and the number of bytes that were sent.
"dsmEndTxn" on page 102	Ends a Storage Manager transaction.
"dsmEndTxnEx" on page 104	Provides group leader object ID information to use with the **dsmGroupHandlerfunction** call.
"dsmGetData" on page 105	Obtains a byte stream of data from Tivoli Storage Manager and place it in the caller's buffer.
"dsmGetBufferData" on page 106	Gets a Tivoli Storage Manager-allocated buffer of data from the Tivoli Storage Manager server.
"dsmGetNextQObj" on page 107	Gets the next query response from a previous **dsmBeginQuery** call and places it in the caller's buffer.
"dsmGetObj" on page 110	Obtains the requested object data from the data stream and places it in the caller's buffer.
"dsmGroupHandler" on page 111	Performs an action on a logical file group depending on the input that is given.
"dsmInit" on page 112	Starts an API session and connects the client to storage.
"**dsmInitEx**" on page 115	Starts an API session using the additional parameters that permit extended verification.
"dsmLogEvent" on page 119	Logs a user message to the server log file, to the local error log, or to both.
"dsmLogEventEx" on page 120	Logs a user message to the server log file, to the local error log, or to both.
"dsmQueryAccess" on page 121	Queries the server for all access authorization rules for either backup versions or archived copies of your objects.
"dsmQueryApiVersion" on page 122	Performs a query request for the API library version that the application client accesses.
"dsmQueryApiVersionEx" on page 122	Performs a query request for the API library version that the application client accesses.
"dsmQueryCliOptions" on page 123	Queries important option values in the user's option files.
"dsmQuerySessInfo" on page 124	Starts a query request to Storage Manager for information that is related to the operation of the specified session in **dsmHandle**.
"dsmQuerySessOptions" on page 125	Queries important option values that are valid in the specified session in **dsmHandle**.
"dsmRCMsg" on page 126	Obtains the message text that is associated with an API return code.
"dsmRegisterFS" on page 127	Registers a new file space with the server.
"dsmReleaseBuffer" on page 128	Returns a Tivoli Storage Manager-allocated buffer.
"dsmRenameObj" on page 128	Renames the high-level or low-level object name.
"dsmRequestBuffer" on page 130	Obtains a Tivoli Storage Manager-allocated buffer for buffer copy elimination.

Table 20. API function calls (continued)

Function call and location	Description
"dsmRetentionEvent" on page 131	Sends a list of object IDs to the server with a retention event operation to be performed on these objects.
"dsmSendBufferData" on page 132	Sends data from a Tivoli Storage Manager-allocated buffer.
"dsmSendData" on page 133	Sends a byte stream of data to Storage Manager via a buffer.
"dsmSendObj" on page 134	Starts a request to send a single object to storage.
"dsmSetAccess" on page 137	Gives other users, or nodes, access to backup versions or archived copies of your objects, access to all your objects, or access to a selective set.
"dsmSetUp" on page 138	Overwrites environment variable values.
"dsmTerminate" on page 140	Ends a session with the server and cleans up the Storage Manager environment.
"dsmUpdateFS" on page 140	Updates a file space in storage.
"dsmUpdateObj" on page 141	Updates the objInfo information that is associated with an active backup object already on the server, or it updates archived objects.
"dsmUpdateObjEx" on page 142	Updates the objInfo information that is associated with a specific archive object even when there are multiple objects with same name, or it updates active backup objects.

Related reference:

API return codes

dsmBeginGetData

The **dsmBeginGetData** function call starts a restore or retrieve operation on a list of objects in storage. This list of objects is contained in the **dsmGetList** structure. The application creates this list with values from the query that preceded a call to **dsmBeginGetData**.

The caller first must use the restore order fields that are obtained from the object query to sort the list that is contained in this call. This ensures that the objects are restored from storage in the most efficient way possible without rewinding or remounting data tapes.

When getting whole objects, the maximum *dsmGetList.numObjID* is DSM_MAX_GET_OBJ. When getting partial objects, the maximum is DSM_MAX_PARTIAL_GET_OBJ.

Follow the call to **dsmBeginGetData** with one or more calls to **dsmGetObj** to obtain each object within the list. After each object is obtained, or additional data for the object is not needed, the **dsmEndGetObj** call is sent.

When all objects are obtained, or the **dsmEndGetObj** is canceled, the **dsmEndGetData** call is sent. You then can start the cycle again.

Syntax

```
dsInt16_t dsmBeginGetData (dsUint32_t       dsmHandle,
   dsBool_t     mountWait,
   dsmGetType   getType,
   dsmGetList  *dsmGetObjListP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsBool_t mountWait (I)
A Boolean true or false value indicates whether or not the application client waits for offline media to be mounted if the data that is needed is currently offline. If mountWait is true, the application waits for the server to mount the required media. The application waits until the media is mounted or the request is canceled.

dsmGetType getType (I)
An enumerated type consisting of **gtBackup** and **gtArchive** that indicates what type of object to get.

dsmGetList *dsmGetObjListP (I)
The structure that contains information about the objects or partial objects to restore or retrieve. The structure points to a list of object IDs and, in the case of a partial object restore or retrieve, a list of associated offsets and lengths. If your application uses the partial object restore or retrieve function, set the **dsmGetList.stVersion** field to **dsmGetListPORVersion**. In a partial object restore or retrieve, you cannot compress data while sending it. To enforce this, set **ObjAttr.objCompressed** to *bTrue*.

See Figure 19 on page 72 and Appendix B, "API type definitions source files," on page 155 for more information on this structure.

See "Partial object restore or retrieve" on page 66 for more information on partial object restore or retrieve.

Return codes

The return code numbers are provided in parentheses ( ).

Table 21. Return codes for dsmBeginGetData

Return code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset that was specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length that was specified during a partial object retrieve is greater than the length of the object, or the offset in addition to the length extends past the end of the object.
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete the request.
DSM_RC_NUMOBJ_EXCEED (2029)	The **dsmGetList.numObjId** is greater than DSM_MAX_GET_OBJ.
DSM_RC_OBJID_NOTFOUND (2063)	The object ID was not found. The object was not restored.
DSM_RC_WRONG_VERSION_PARM (2065)	The API version of the application client is different from the Tivoli Storage Manager library version.

dsmBeginQuery

The **dsmBeginQuery** function call starts a query request to the server for information about data, file spaces, and management classes.

Specifically, **dsmBeginQuery** can query:
- Archived data
- Backed-up data
- Active backed-up data
- File spaces
- Management classes

The query data that is returned from the call is obtained by one or more calls to **dsmGetNextQObj**. When the query is complete, the **dsmEndQuery** call is sent.

Syntax

```
dsInt16_t dsmBeginQuery  (dsUint32_t       dsmHandle,
   dsmQueryType  queryType,
   dsmQueryBuff  *queryBuffer);
```

Parameters

dsUint32_t dsmHandle (I)

The handle that associates this call with a previous **dsmInitEx** call.

dsmQueryType queryType (I)

Identifies the type of query to run. Assign one of the following options:

qtArchive

Queries archived objects.

qtBackup

Queries backed-up objects.

qtBackupActive

Queries active, backed-up objects only for the entire file space name that you pass. This query is called a "fast path" and is an efficient way to query active objects from storage.

Prerequisite: You must be logged on as a root user on a UNIX or Linux operating system.

qtFilespace

Queries registered file spaces.

qtMC

Queries defined management classes.

qtBackupGroups

Queries groups that are closed.

qtOpenGroups

Queries groups that are open.

qtProxyNodeAuth

Queries nodes to which this node can proxy.

qtProxyNodePeer

Queries peer nodes with the same target.

dsmQueryBuff *queryBuffer (I)

Identifies a pointer to a buffer that is mapped to a particular data structure. This structure is associated with the query type that you pass. These structures contain the selection criteria for each query type. Complete the fields in each structure to specify the scope of the query that you want to run. The `stVersion` field in each structure contains the structure version number.

The data structures and their related fields include the following items:

qryArchiveData

objName

The complete object name. You can use a wildcard character, such as an asterisk (*) or a question mark (?), in the high-level or low-level portion of the name. An asterisk matches zero or more characters, and a question mark matches one character. The `objType` field of `objName` can have one of the following values:

- `DSM_OBJ_FILE`
- `DSM_OBJ_DIRECTORY`
- `DSM_OBJ_ANY_TYPE`

For more information about high-level and low-level names, see the following topic: "High-level and low-level names" on page 24.

owner

The owner name of the object.

insDateLowerBound

The lower boundary for the insert date that the object was archived. To obtain the default lower boundary, set the year component to `DATE_MINUS_INFINITE`.

insDateUpperBound

The upper boundary for the insert date that the object was archived. To obtain the default upper boundary, set the year component to `DATE_PLUS_INFINITE`.

expDateLowerBound

The lower boundary for the expiration date. The default values for both expiration date fields are the same as for the insert date fields.

expDateUpperBound

The upper boundary for the expiration date.

descr

The archive description. Enter an asterisk (*) to search all descriptions.

qryBackupData

objName

The complete object name. You can use a wildcard character, such as an asterisk (*) or a question mark (?), in the high-level or low-level portion of the name. An asterisk matches zero or more characters, and a question mark matches one character. The objType field of objName can have one of the following values:

- DSM_OBJ_FILE
- DSM_OBJ_DIRECTORY
- DSM_OBJ_ANY_TYPE

For more information about high-level and low-level names, see the following topic: "High-level and low-level names" on page 24.

owner

The owner name of the object.

objState

You can query for one of the following object states:

- DSM_ACTIVE
- DSM_INACTIVE
- DSM_ANY_MATCH

pitDate

The point-in-time value. A query with this field returns the most recent object that is backed up before this date and time. The objState can be active or inactive. Objects that are deleted before the pitDate are not returned. For example:

```
Mon - backup ABC(1), DEF, GHI
Tue - backup ABC(2), delete DEF
Thr - backup ABC(3)
```

On Friday, call the query with a point-in-time value of Wednesday at 12:00:00 a.m. The call returns the following information:

```
ABC(2) - an Inactive copy
GHI    - an Active copy
```

The call does not return DEF because that object as deleted prior to the point-in-time value.

qryABackupData

objName

The complete object name. You can use a wildcard character, such as an asterisk (*) or a question mark (?), in the high-level or low-level portion of the name. An asterisk matches zero or more characters, and a question mark matches one character. The objType field of objName can have one of the following values:

- DSM_OBJ_FILE
- DSM_OBJ_DIRECTORY
- DSM_OBJ_ANY_TYPE

For more information about high-level and low-level names, see the following topic: "High-level and low-level names" on page 24.

qryFSData

fsName

Enter the name of a specific file space in this field, or enter an asterisk (*) to retrieve information about all registered file spaces.

qryMCData

mcName

Enter the name of a specific management class, or enter an empty string (" ") to retrieve information about all management classes.

Note: You cannot use an asterisk (*).

mcDetail

Determines whether information on the backup and archive copy groups of the management class is returned. The following values are valid:

- `bTrue`
- `bFalse`

qryBackupGroup:

groupType

The group type is `DSM_GROUPTYPE_PEER`.

fsName

The file space name.

owner

The owner ID.

groupLeaderObjId

The group leader object ID.

objType

The object type.

qryProxyNodeAuth:

targetNodeName

The target node name.

peerNodeName

The peer node name.

hlAddress

The peer address of the high-level name.

llAddress

The peer address of the low-level name.

qryProxyNodePeer:

targetNodeName

The target node name.

peerNodeName

The peer node name.

hlAddress

The peer address of the high-level name.

llAddress

The peer address of the low-level name.

Return codes

The following table describes the return codes for the **dsmBeginQuery** function call.

Table 22. Return codes for dsmBeginQuery

Return code	Return code number	Explanation
DSM_RC_NO_MEMORY	102	There is not enough memory to complete the request.
DSM_RC_FILE_SPACE_NOT_FOUND	124	The specified file space was not found.
DSM_RC_NO_POLICY_BLK	2007	Server policy information was not available.
DSM_RC_INVALID_OBJTYPE	2010	Invalid object type.
DSM_RC_INVALID_OBJOWNER	2019	Invalid object owner name.
DSM_RC_INVALID_OBJSTATE	2024	Invalid object condition.
DSM_RC_WRONG_VERSION_PARM	2065	The API version of the application client is different from the Tivoli Storage Manager library version.

dsmBeginTxn

The **dsmBeginTxn** function call begins one or more IBM Tivoli Storage Manager transactions that begin a complete action; either all the actions succeed or none succeed. An action can be either a single call or a series of calls. For example, a **dsmSendObj** call that is followed by a number of **dsmSendData** calls can be considered a single action. Similarly, a **dsmSendObj** call with a **dataBlkPtr** that indicates a data area containing the object to back up is also considered a single action.

Try to group more than one object together in a single transaction for data transfer operations. Grouping objects results in significant performance improvements in the Tivoli Storage Manager system. From both a client and a server perspective, a certain amount of overhead is incurred by starting and ending each transaction.

There are limits to what you can perform within a single transaction. These restrictions include:

- A maximum number of objects that you can send or delete in a single transaction. This limit is located in the data that **dsmQuerySessInfo** returns in the *ApiSessInfo.maxObjPerTxn* field. This corresponds to the *TxnGroupMax* server option.
- All objects that are sent to the server (either backup or archive) within a single transaction must have the same copy destination that is defined in the

management class binding for the object. This value is located in the data that **dsmBindMC** returns in the **mcBindKey.backup_copy_dest** or **mcBindKey.archive_copy_dest** fields.

With the API, either the application client can monitor and control these restrictions, or the API can monitor these restrictions. If the API is monitoring restrictions, appropriate return codes from the API calls inform the application client when one or more restrictions are reached.

Always match a **dsmBeginTxn** call with a **dsmEndTxn** call to optimize the set of actions within a pair of **dsmBeginTxn** and **dsmEndTxn** calls.

Syntax

```
dsInt16_t dsmBeginTxn  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 23. Return codes for dsmBeginTxn

Return code	Explanation
DSM_RC_ABORT_NODE_NOT_AUTHORIZED (36)	FROMNODE or FROMOWNER is not allowed for TXN operations.

dsmBindMC

The **dsmBindMC** function call associates, or binds, a management class to the passed object. The object is passed through the include-exclude list that is pointed to in the options file. If a match is not found in the Include list for a specific management class, the default management class is assigned. The Exclude list can prevent objects from a backup but not from an archive.

The application client can use the parameters that are returned in the `mcBindKey` structure to determine if this object should be backed up or archived, or whether a new transaction must be started because of different copy destinations. See **dsmBeginTxn** for more information.

Call **dsmBindMC** before you call **dsmSendObj** because every object must have a management class associated with it. This call can be performed within a transaction or outside of a transaction. For example, within a multiple object transaction, if **dsmBindMC** indicates that the object has a different copy destination than the previous object, the transaction must be ended and a new transaction started. In this case, another **dsmBindMC** is not required because one has already been performed for this object.

Syntax

```
dsInt16_t dsmBindMC  (dsUint32_t       dsmHandle,
   dsmObjName  *objNameP,
   dsmSendType  sendType,
   mcBindKey   *mcBindKeyP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmObjName *objNameP (I)
A pointer to the structure that contains the file space name, high-level object name, low-level object name, and object type.

dsmSendType sendType (I)
Identifies whether this management class bind is performed for archive or backup sends. The possible values for this call include:

Name	Description
stBackup	A backup object
stArchive	An archive object
stBackupMountWait	A backup object
stArchiveMountWait	An archive object

For the **dsmBindMC** call, stBackup and stBackupMountWait are equivalent, and stArchive and stArchiveMountWait are equivalent.

mcBindKey *mcBindKeyP (O)
This is the address of an mcBindKey structure where the management class information is returned. The application client can use the information that is returned here to determine if this object fits within a multiple object transaction, or to perform a management class query on the management class that is bound to the object.

Return codes

The return code numbers are provided in parentheses ( ).

Table 24. Return codes for dsmBindMC

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete the request.
DSM_RC_INVALID_PARM (109)	One of the parameters that was passed has an invalid value.
DSM_RC_TL_EXCLUDED (185)	The backup object is excluded and cannot be sent.
DSM_RC_INVALID_OBJTYPE (2010)	Invalid object type.
DSM_RC_INVALID_SENDTYPE (2022)	Invalid send type.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client API version is different from the Tivoli Storage Manager library version.

dsmChangePW

The **dsmChangePW** function call changes an IBM Tivoli Storage Manager password. On a multiple-user operating system such as UNIX or Linux, only the root user or the authorized user can use this call.

On Windows operating systems, you can specify the password in the dsm.opt file. In this situation, **dsmChangePW** does not update the dsm.opt file. After the call to **dsmChangePW** is made, you must update the dsm.opt file separately.

This call must process successfully if **dsmInitEx** returns DSM_RC_VERIFIER_EXPIRED. The session ends if the **dsmChangePW** call fails in this situation.

If **dsmChangePW** is called for some other reason, the session remains open regardless of the return code.

Syntax

```
dsInt16_t dsmChangePW  (dsUint32_t   dsmHandle,
   char    *oldPW,
   char    *newPW);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

char *oldPW (I)
The old password of the caller. The maximum length is DSM_MAX_VERIFIER_LENGTH.

char *newPW (I)
The new password of the caller. The maximum length is DSM_MAX_VERIFIER_LENGTH.

Return codes

The return code numbers are provided in parentheses ( ).

Table 25. Return codes for dsmChangePW

Return code	Explanation
DSM_RC_ABORT_BAD_VERIFIER (6)	An incorrect password was entered.
DSM_RC_AUTH_FAILURE (137)	Authentication failure. Old password is incorrect.
DSM_RC_NEWPW_REQD (2030)	A value must be entered for the new password.
DSM_RC_OLDPW_REQD (2031)	A value must be entered for the old password.
DSM_RC_PASSWD_TOOLONG (2103)	The specified password is too long.
DSM_RC_NEED_ROOT (2300)	The API caller must be a root user or an authorized user.

dsmCleanUp

The **dsmCleanUp** function call is used if **dsmSetUp** was called. The **dsmCleanUp** function call should be called after **dsmTerminate**. You cannot make any other calls after you call **dsmCleanUp**.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t DSMLINKAGE dsmCleanUp
    (dsBool_t       mtFlag);
```

Parameters

dsBool_t mtFlag (I)
This parameter specifies that the API was used either in a single thread or a multithread mode. Possible values include:

- DSM_SINGLETHREAD
- DSM_MULTITHREAD

dsmDeleteAccess

The **dsmDeleteAccess** function call deletes current authorization rules for backup versions or archived copies of your objects. When you delete an authorization rule, you revoke the access a user has to any files that are specified by the rule.

When you use **dsmDeleteAccess**, you can only delete one rule at a time. Obtain the rule ID through the **dsmQueryAccess** command.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t DSMLINKAGE dsmDeleteAccess
             (dsUint32_t          dsmHandle,
              dsUint32_t          ruleNum) ;
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsUint32_t ruleNum (I)
The rule ID for the access rule that is deleted. This value is obtained from a **dsmQueryAccess** function call.

dsmDeleteFS

The **dsmDeleteFS** function call deletes a file space from storage. To delete a file space, you must have the appropriate permissions that your Tivoli Storage Manager administrator gave you. To determine whether you have the necessary permissions, call **dsmQuerySessInfo**. This function call returns a data structure of type *ApiSessInfo*, that includes two fields, *archDel* and *backDel*.

Note:

- On a UNIX or Linux operating system, only a root user or an authorized user can delete a file space.
- If the file space that you need to delete contains backup versions, you must have backup delete authority (***backDel*** = BACKDEL_YES). If the file space contains archive copies, you must have archive delete authority (*archDel* = ARCHDEL_YES). If the file space contains both backup versions and archive copies, you must have both types of delete authority.
- When using an archive manager server, a file space cannot actually be removed. This function call returns *rc=0* even though the file space was not actually deleted. The only way to verify that the file space has been deleted is to issue a filespace query to the server.
- The Tivoli Storage Manager server delete file-space function is a background process. If errors other than those detected before passing a return code happen, they are recorded in the Tivoli Storage Manager server log.

Syntax

```
dsInt16_t dsmDeleteFS  (dsUint32_t          dsmHandle,
   char           *fsName,
   unsigned char   repository);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

char *fsName (I)
A pointer to the file space name to delete. The wildcard character is not permitted.

unsigned char repository (I)
Indicates whether the file space to delete is a backup repository, archive repository, or both. The possible values for this field include:

```
DSM_ARCHIVE_REP     /* archive repository   */
DSM_BACKUP_REP      /* backup repository    */
DSM_REPOS_ALL       /* all repository types */
```

Return codes

The return code numbers are provided in parentheses ( ).

Table 26. Return codes for dsmDeleteFS

Return code	Explanation
DSM_RC_ABORT_NOT_AUTHORIZED (27)	You do not have the necessary authority to delete the file space.
DSM_RC_INVALID_REPOS (2015)	Invalid value for repository.
DSM_RC_FSNAME_NOTFOUND (2060)	File space name not found.
DSM_RC_NEED_ROOT (2300)	API caller must be a root user.

dsmDeleteObj

The **dsmDeleteObj** function call inactivates backup objects, deletes backup objects, or it deletes archive objects in storage. The **dtBackup** type inactivates the currently active backup copy only. The **dtBackupID** type removes from the server whichever object ID is specified. Call this function from within a transaction.

See **dsmBeginTxn** for more information.

Before you send **dsmDeleteObj**, send the query sequence that is described in "Querying the Tivoli Storage Manager system" on page 33 to obtain the information for **delInfo**. The call to **dsmGetNextQObj** returns a data structure named **qryRespBackupData** for backup queries or **qryRespArchiveData** for archive queries. These data structures contain the information that you need for **delInfo**.

The value of **maxObjPerTxn** determines the maximum number of objects that you can delete in a single transaction. To obtain this value, call **dsmQuerySessInfo**.

Tip: Your node must have the appropriate permission that your administrator set. To delete archive objects, you must have archive delete authority. You do not need backup delete authority to inactivate a backup object.

Syntax

```
dsInt16_t dsmDeleteObj  (dsUint32_t      dsmHandle,
   dsmDelType  delType,
   dsmDelInfo  delInfo)
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmDelType delType (I)
Indicates what type of object (backup or archive) to delete. Possible values include:

Name	Description
dtArchive	The object to delete was previously archived.
dtBackup	The object to inactivate was previously backed up.
dtBackupID	The object to delete was previously backed up. **Restriction:** Using this **delType** with *objID* removes the backup object from the server. Only an owner of an object can delete it. You can delete any version (active or inactive) of an object. The server reconciles the versions. If you delete an active version of an object, the first inactive version becomes active. If you delete an inactive version of an object, all older versions will advance. The node must be registered with **backDel** permission.

dsmDelInfo delInfo (I)
A structure whose fields identify the object. The fields are different, depending on whether the object is a backup object or an archive object. The structure to inactivate a backup object, `delBack`, contains the object name and the object copy group. The structure for an archive object, `delArch`, contains the object ID.

The structure to remove a backup object, `delBackID`, contains the object ID.

Return codes

The return code numbers are provided in parentheses ( ).

Table 27. Return codes for dsmDeleteObj

Return code	Explanation
DSM_RC_FS_NOT_REGISTERED (2061)	File space name is not registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client API version is different from the Tivoli Storage Manager library version.

dsmEndGetData

The **dsmEndGetData** function call ends a **dsmBeginGetData** session that obtains objects from storage.

The **dsmEndGetData** function call starts after all objects that you want to restore are processed, or ends the get process prematurely. Call **dsmEndGetData** to end a **dsmBeginGetData** session before you can continue other processing.

Depending on when **dsmEndGetData** is called, the API might need to finish processing a partial data stream before the process can be stopped. The caller, therefore, should not expect an immediate return from this call. Use **dsmTerminate** if the application needs to close the session and end the restore immediately.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmEndGetData  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmEndGetDataEx

The **dsmEndGetDataEx** function call provides the total of LAN-free bytes that were sent. It is an extension of the **dsmEndGetData** function call.

Syntax

There are no return codes that are specific to this call.

```
dsInt16_t dsmEndGetDataEx (dsmEndGetDataExIn_t * dsmEndGetDataExInP,
                           dsmEndGetDataExOut_t * dsmEndGetDataExOutP);
```

Parameters

dsmEndGetDataExIn_t *dsmEndGetDataExInP (I)
Passes the end get object dsmHandle that identifies the session and associates it with subsequent calls.

dsmEndGetDataExOut_t *dsmEndGetDataExOutP (O)
This structure contains this input parameter:

totalLFBytesRecv
The total LAN-free bytes that are received.

dsmEndGetObj

The **dsmEndGetObj** function call ends a **dsmGetObj** session that obtains data for a specified object.

Start the **dsmEndGetObj** call after an end of data is received for the object. This indicates that all data was received, or that no more data will be received for this object. Before you can start another **dsmGetObj** call, you must call **dsmEndGetObj**.

Depending on when **dsmEndGetObj** is called, the API might need to finish processing a partial data stream before the process can stop. Do not expect an immediate return from this call.

Syntax

```
dsInt16_t dsmEndGetObj  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 28. Return codes for dsmEndGetObj

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete the request.

dsmEndQuery

The **dsmEndQuery** function call signifies the end of a **dsmBeginQuery** action. The application client sends **dsmEndQuery** to complete a query. This call either is sent after all query responses are obtained through **dsmGetNextQObj**, or it is sent to end a query before all data are returned.

Tip: Tivoli Storage Manager continues to send the query data from the server to the client in this case, but the API discards any remaining data.

Once a **dsmBeginQuery** is sent, a **dsmEndQuery** must be sent before any other activity can start.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmEndQuery  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmEndSendObj

The **dsmEndSendObj** function call indicates the end of data that is sent to storage.

Enter the **dsmEndSendObj** function call to indicate the end of data from the **dsmSendObj** and **dsmSendData** calls. A protocol violation occurs if this is not performed. The exception to this rule is if you call **dsmEndTxn** to end the transaction. Doing this discards all data that was sent for the transaction.

Syntax

```
dsInt16_t dsmEndSendObj  (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 29. Return codes for dsmEndSendObj

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete this request.

dsmEndSendObjEx

The **dsmEndSendObjEx** function call provides additional information regarding the number of bytes processed. The information includes: total bytes sent, compression information, lan-free bytes, and deduplication information.

The **dsmEndSendObjEx** function call is an extension of the **dsmEndSendObj** function call.

Syntax

```
dsInt16_t dsmEndSendObjEx (dsmEndSendObjExIn_t  *dsmEndSendObjExInP,
                           dsmEndSendObjExOut_t *dsmEndSendObjExOutP);
```

Parameters

dsmEndSendObjExIn_t *dsmEndSendObjExInP (I)
This parameter passes the end send object dsmHandle that identifies the session and associates it with subsequent calls.

dsmEndSendObjExOut_t *dsmEndSendObjExOutP (O)
This parameter passes the end send object information:

Name	Description
totalBytesSent	The total number of bytes that are read from the application.
objCompressed	A flag that displays if the object was compressed.
totalCompressedSize	The total byte size after compression.
totalLFBytesSent	The total LAN-free bytes that were sent.
objDeduplicated	A flag that displays if the object was deduplicated by the API.
totalDedupSize	Total bytes sent after deduplication.

Return codes

The return code numbers are provided in parentheses ( ).

Table 30. Return codes for dsmEndSendObjEx

Return code	Explanation
DSM_RC_NO_MEMORY (102)	There is no RAM remaining to complete this request.

dsmEndTxn

The **dsmEndTxn** function call ends an IBM Tivoli Storage Manager transaction. Pair the **dsmEndTxn** function call with **dsmBeginTxn** to identify the call or set of calls that are considered a transaction. The application client can specify on the **dsmEndTxn** call whether the transaction must be committed or ended.

Perform all of the following calls within the bounds of a transaction:

- **dsmSendObj**
- **dsmSendData**
- **dsmEndSendObj**
- **dsmDeleteObj**

Syntax

```
dsInt16_t dsmEndTxn (dsUint32_t   dsmHandle,
   dsUint8_t    vote,
   dsUint16_t  *reason);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsUint8_t vote (I)
Indicates whether the application client commits all the actions that are done between the previous **dsmBeginTxn** call and this call. The following values are possible:

```
DSM_VOTE_COMMIT     /* commit current transaction    */
DSM_VOTE_ABORT      /* roll back current transaction */
```

Use DSM_VOTE_ABORT only if your application finds a reason to stop the transaction.

dsUint16_t *reason (O)
If the call to **dsmEndTxn** ends with an error, or the value of vote is not agreed to, this parameter has a reason code that indicates why the vote failed. The return code for the call might be zero, and the reason code might be non-zero. Therefore, the application client must always check for errors on both the return code and the reason (if (rc || reason)) before you can assume a successful completion.

If the application specifies a vote of DSM_VOTE_ABORT, the reason code is DSM_RS_ABORT_BY_CLIENT (3). See Appendix A, "API return codes source file: dsmrc.h," on page 145 for a list of the possible reason codes. Numbers 1 through 50 in the return codes list are reserved for the reason codes. If the server ends the transaction, the return code is DSM_RC_CHECK_REASON_CODE. In this case, the reason value contains more information on the cause of the abort.

Return codes

The return code numbers are provided in parentheses ( ).

Table 31. Return codes for dsmEndTxn

Return code	Explanation
DSM_RC_ABORT_CRC_FAILED (236)	The CRC that was received from the server does not match the CRC that was calculated by the client.
DSM_RC_INVALID_VOTE (2011)	The value that was specified for vote is not valid.
DSM_RC_CHECK_REASON_CODE (2302)	The transaction was aborted. Check the reason field.
DSM_RC_ABORT_STGPOOL_COPY_CONT_NO (241)	The write to one of the copy storage pools failed, and the Tivoli Storage Manager storage pool option COPYCONTINUE is set to NO. The transaction terminates.
DSM_RC_ABORT_RETRY_SINGLE_TXN (242)	This abort code indicates that the current transaction was aborted because of a problem during a store operation. The problem can be resolved by sending each file in an individual transaction. This error is typical in the following circumstances: • The next storage pool has a different copy storage pool list. • The operation is switched to this pool in the middle of a transaction. .

dsmEndTxnEx

The **dsmEndTxnEx** function call provides group leader object ID information for you to use with the **dsmGroupHandler** function call. It is an extension of the **dsmEndTxn** function call.

Syntax

```
dsInt16_t dsmEndTxnEx (dsmEndTxnExIn_t *dsmEndTxnExInP
                       dsmEndTxnExOut_t *dsmEndTxnExOutP);
```

Parameters

dsmEndTxnExIn_t *dsmEndTxnExInP (I)
This structure contains the following parameters:

dsmHandle
The handle that identifies the session and associates it with subsequent Tivoli Storage Manager calls.

dsUint8_t vote (I)
Indicates whether or not the application client commits all the actions that are done between the previous **dsmBeginTxn** call and this call. The possible values are:

```
DSM_VOTE_COMMIT     /* commit current transaction    */
DSM_VOTE_ABORT      /* roll back current transaction */
```

Use DSM_VOTE_ABORT only if your application has found a reason to stop the transaction.

dsmEndTxnExOut_t *dsmEndTxnExOutP (O)
This structure contains the following parameters:

dsUint16_t *reason (O)
If the call to **dsmEndTxnEx** ends with an error or the value of *vote* is not agreed to, this parameter has a reason code indicating why the vote failed.

Tip: The return code for the call might be zero, and the reason code might be non-zero. Therefore, the application client must always check for errors on both the return code and the reason (`if (rc || reason)`) before you can assume a successful completion.

If the application specifies a vote of DSM_VOTE_ABORT, the reason code is DSM_RS_ABORT_BY_CLIENT (3). See Appendix A, "API return codes source file: dsmrc.h," on page 145 for a list of the possible reason codes. Numbers 1 through 50 in the return codes list are reserved for the reason codes. If the server ends the transaction, the return code is DSM_RC_CHECK_REASON_CODE. In this case, the reason value contains more information on the cause of the abort.

groupLeaderObjId
The group leader object ID that is returned when the DSM_ACTION_OPEN flag is used with the **dsmGroupHandler** call.

Return codes

The return code numbers are provided in parentheses ( ).

Table 32. Return codes for dsmEndTxnEx

Return code	Explanation
DSM_RC_INVALID_VOTE (2011)	The value that was specified for vote is invalid.
DSM_RC_CHECK_REASON_CODE (2302)	The transaction was aborted. Check the reason field.
DSM_RC_ABORT_STGPOOL_COPY_CONT_NO (241)	The write to one of the copy storage pools failed, and the Tivoli Storage Manager storage pool option COPYCONTINUE was set to NO. The transaction terminates.
DSM_RC_ABORT_RETRY_SINGLE_TXN (242)	During a simultaneous-write operation, an object in the transaction is going to a destination with different copy storage pools. End the current transaction and send each object again in its own transaction.

dsmGetData

The **dsmGetData** function call obtains a byte stream of data from Tivoli Storage Manager and places it in the caller's buffer. The application client calls **dsmGetData** when there is more data to receive from a previous **dsmGetObj** or **dsmGetData** call.

Syntax

```
dsInt16_t dsmGetData  (dsUint32_t   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

DataBlk *dataBlkPtr (I/O)
Points to a structure that includes both a pointer to the buffer for the data that is received and the size of the buffer. On return, this structure contains the number of bytes that is actually transferred. See Appendix B, "API type definitions source files," on page 155 for the type definition.

Return codes

The return code numbers are provided in parentheses ( ).

Table 33. Return codes for dsmGetData

Return code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset that was specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length that was specified during a partial object retrieve is greater than the length of the object, or the offset in addition to the length extends beyond the end of the object.
DSM_RC_FINISHED (121)	Finished processing. The last buffer was received. Check `numBytes` for the amount of data and then call Tivoli Storage ManagerdsmEndGetObj.
DSM_RC_NULL_DATABLKPTR (2001)	Datablock pointer is null.
DSM_RC_ZERO_BUFLEN (2008)	Buffer length is zero for datablock pointer.
DSM_RC_NULL_BUFPTR (2009)	Buffer pointer is null for datablock pointer.

Table 33. Return codes for dsmGetData (continued)

Return code	Explanation
DSM_RC_WRONG_VERSION_PARM (2065)	The application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_MORE_DATA (2200)	There is more data to get.

dsmGetBufferData

The **dsmGetBufferData** function call receives a byte stream of data from IBM Tivoli Storage Manager through a buffer. After each call the application needs to copy the data and release the buffer through a call to **dsmReleaseBuffer**. If the number of buffers held by the application equals the numTsmBuffers specified in the **dsmInitEx** call, the **dsmGetBufferData** function blocks until a **dsmReleaseBuffer** is called.

Syntax

```
dsInt16_t  dsmGetBufferData   (getDatatExIn_t        *dsmGetBufferDataExInP,
                               getDataExOut_t        *dsmGetBufferDataExOutP) ;
```

Parameters

getDataExIn_t * dsmGetBufferDataExInP (I)
This structure contains the following input parameter.

dsUint32_t dsmHandle
The handle that identifies the session and associates it with a previous **dsmInitEx** call.

getDataExOut_t * dsmGetBufferDataExOutP (0)
This structure contains the following output parameters.

dsUint8_t tsmBufferHandle(0)
The handle that identifies the buffer received.

char *dataPtr(0)
The address to which the data was written.

dsUint32_t numBytes(0)
Actual number of bytes written by Tivoli Storage Manager.

Return codes

The return code numbers are provided in parentheses ( ).

Table 34. Return codes for dsmGetBufferData

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE (2041)	The call was not issued in the proper state.
DSM_RC_OBJ_ENCRYPTED (2049)	This function cannot be used for encrypted objects.
DSM_RC_OBJ_COMPRESSED (2048)	This function cannot be used for compressed objects.
DSM_RC_BUFF_ARRAY_ERROR (2045)	A buffer array error occurred.

dsmGetNextQObj

The **dsmGetNextQObj** function call gets the next query response from a previous **dsmBeginQuery** call and places the response in the caller buffer.

The **dsmGetNextQObj** call is called one or more times. Each time the function is called, either a single query record is retrieved, or an error or a DSM_RC_FINISHED reason code is returned. If DSM_RC_FINISHED is returned, there is no more data to process. When all query data is retrieved, or if no more query data is needed, send the **dsmEndQuery** call to end the query process.

The **dataBlkPtr** parameter must point to a buffer that is defined with the **qryResp*Data** structure type. The context in which **dsmGetNextQObj** is called determines the type of structure that is entered on the query response.

Syntax

```
dsInt16_t dsmGetNextQObj (dsUint32_t   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

DataBlk *dataBlkPtr (I/O)
Points to a structure that includes both a pointer to the buffer for the data to be received and the size of the buffer. This buffer is the **qryResp*Data** response structure. On return, this structure contains the number of bytes that is transferred. The structure that is associated with each type of query is described in the following table. For more information about the type definition of **DataBlk**, see the following topic: Appendix B, "API type definitions source files," on page 155.

Table 35. DataBlk pointer structure

Query	Response structure	Fields of special interest
qtArchive	**qryRespArchiveData**	**sizeEstimate** Contains the value that is passed on a previous **dsmSendObj** call. **mediaClass** Can have a value of MEDIA_FIXED if the object is on disk, or MEDIA_LIBRARY if the object is on tape. **clientDeduplicated** Indicates whether this object is deduplicated by the client.

Table 35. DataBlk pointer structure (continued)

Query	Response structure	Fields of special interest
qtBackup	**qryRespBackupData**	**restoreOrderExt** Is of type dsUint16_t. Sort on this field when several objects are restored on a **dsmBeginGetData** call. An example of sorting code for this call is in the API sample, dapiqry.c. For a sorting example, see the following topic: Figure 16 on page 68. **sizeEstimate** Contains the value that is passed on a previous **dsmSendObj** call. **mediaClass** Can have a value of MEDIA_FIXED if the object is on disk or MEDIA_LIBRARY if the object is on tape. **clientDeduplicated** Indicates whether this object is deduplicated by the client.
qtBackupActive	**qryARespBackupData**	
qtBackupGroups	**qryRespBackupData**	**dsBool_t isGroupLeader** If true, signifies this object is a group leader.
qtOpenGroups	**qryRespBackupData**	**dsBool_t isOpenGroup;** If true, signifies this group is open and not complete.

Table 35. DataBlk pointer structure (continued)

Query	Response structure	Fields of special interest
qtFilespace	**qryRespFSData**	**backStartDate** Contains the server time stamp when the file space is updated with the **backStartDate** action. **backCompleteDate** Contains the server time stamp when the file space is updated with the **backCompleteDate** action. **lastReplStartDate** Contains the time stamp for the last time that replication was started on the server. **lastReplCmpltDate** Contains the time stamp for the last time that replication was completed, even if there was a failure. **lastBackOpDateFromServer** Contains the last store time stamp that was saved on the server. **lastBackOpDateFromLocal** Contains the last store time stamp that was saved on the client.
qtMC	**qryRespMCData** **qryRespMCDetailData**	
qtProxyNodeAuth	**qryRespProxyNodeData** **targetNodeName** **peerNodeName** **hlAddress** **llAddress**	
qtProxyNodePeer	**qryRespProaxyNodeData** **targetNodeName** **peerNodeName** **hlAddress** **llAddress**	

Return codes

The following table describes the return codes for the **dsmGetNextQObj** function call.

Table 36. Return codes for the **dsmGetNextQObj** *function call*

Return code	Return code number	Description
DSM_RC_ABORT_NO_MATCH	2	No match for the query was requested.
DSM_RC_FINISHED	121	Finished processing (start **dsmEndQuery**). There is no more data to process.
DSM_RC_UNKNOWN_FORMAT	122	The file that Tivoli Storage Manager attempted to restore or retrieve has an unknown format.

Table 36. Return codes for the **dsmGetNextQObj** *function call (continued)*

Return code	Return code number	Description
DSM_RC_COMM_PROTOCOL_ERROR	136	Communication protocol error.
DSM_RC_NULL_DATABLKPTR	2001	Pointer is not pointing to a data block.
DSM_RC_INVALID_MCNAME	2025	Invalid management class name.
DSM_RC_BAD_CALL_SEQUENCE	2041	The sequence of calls is invalid.
DSM_RC_WRONG_VERSION_PARM	2065	The version of the application client API is different from the Tivoli Storage Manager library version.
DSM_RC_MORE_DATA	2200	There is more data to get.
DSM_RC_BUFF_TOO_SMALL	2210	Buffer is too small.

dsmGetObj

The **dsmGetObj** function call obtains the requested object data from the Tivoli Storage Manager data stream and places it in the caller's buffer. The **dsmGetObj** call uses the object ID to obtain the next object or partial object from the data stream.

The data for the indicated object is placed in the buffer to which **DataBlk** points. If more data is available, you must make one or more calls to **dsmGetData** to receive the remaining object data until a return code of DSM_RC_FINISHED is returned. Check the numBytes field in **DataBlk** to see whether any data remains in the buffer.

Objects should be asked for in the order that they were listed on the **dsmBeginGetData** call in the **dsmGetList** parameter. The exception is when the application client needs to pass over an object in the data stream to get to an object later in the list. If the object that is indicated by the object ID is not the next object in the stream, the data stream is processed until the object is located, or the stream is completed. Use this feature with care, because it might be necessary to process and discard large amounts of data to locate the requested object.

Requirement: If **dsmGetObj** returns a failure code (NOT FINISHED or MORE_DATA), the session must be terminated to stop the restore operation. This is especially important when you use encryption and receive a RC_ENC_WRONG_KEY. You must start a new session with the proper key.

Syntax

```
dsInt16_t dsmGetObj (dsUint32_t dsmHandle,
   ObjID    *objIdP,
   DataBlk  *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

ObjID *objIdP (I)
A pointer to the ID of the object to restore.

DataBlk *dataBlkPtr (I/O)
A pointer to the buffer where the restored data are placed.

Return codes

The return code numbers are provided in parentheses ( ).

Table 37. Return codes for dsmGetObj

Return code	Explanation
DSM_RC_ABORT_INVALID_OFFSET (33)	The offset that is specified during a partial object retrieve is greater than the length of the object.
DSM_RC_ABORT_INVALID_LENGTH (34)	The length that is specified during a partial object retrieve is greater than the length of the object, or the offset in addition to the length extends past the end of the object.
DSM_RC_FINISHED (121)	Finished processing (start **dsmEndGetObj**).
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_MORE_DATA (2200)	There is more data to get.
RC_ENC_WRONG_KEY (4580)	The key provided in the **dsmInitEx** call, or the saved key, does not match the key that was used to encrypt this object. Terminate the session and provide the proper key.

dsmGroupHandler

The **dsmGroupHandler** function call performs an action on a logical file group depending on the input that is given. The client relates a number of individual objects together to reference and manage on the Tivoli Storage Manager server as a logical group.

For more information, see "File grouping" on page 63.

Syntax

```
dsInt16_t dsmGroupHandler (dsmGroupHandlerIn_t   *dsmGroupHandlerInP,
                           dsmGroupHandlerOut_t  *dsmGroupHandlerOutP);
```

Parameters

dsmGroupHandlerIn_t *dsmGroupHandlerInP (I)
Passes group attributes to the API.

groupType
The type of the group. Values include:
- DSM_GROUPTYPE_PEER - peer group

actionType
The action to be executed. Values include:
- DSM_GROUP_ACTION_OPEN - creates a new group
- DSM_GROUP_ACTION_CLOSE - commits and saves an open group
- DSM_GROUP_ACTION_ADD - appends to a group
- DSM_GROUP_ACTION_ASSIGNTO - assigns to another group
- DSM_GROUP_ACTION_REMOVE- removes a member from a group

memberType.
The group type of the object. Values include:
- DSM_MEMBERTYPE_LEADER - group leader
- DSM_MEMBERTYPE_MEMBER - group member

***uniqueGroupTagP**
A unique string ID that is associated with a group.

leaderObjId
The Object ID for the group leader.

***objNameP**
A pointer to the object name of the group leader.

memberObjList
A list of objects to remove or assign.

dsmGroupHandlerOut_t *dsmGroupHandlerOutP (O)
Passes the address of the structure that the API completes. The structure version number is returned.

Return codes

The return code numbers are provided in parentheses ( ).

Table 38. Return codes for dsmGroupHandler

Return code	Explanation
DSM_RC_ABORT_INVALID_GROUP_ACTION (237)	An invalid operation was attempted on a group leader or member.

dsmInit

The **dsmInit** function call starts an API session and connects the client to Tivoli Storage Manager storage. The application client can have only one active session open at a time. To open another session with different parameters, use the **dsmTerminate** call first to end the current session.

To permit cross-node query and restore or retrieve, use the *-fromnode* and *-fromowner* string options. See "Accessing objects across nodes and owners" on page 25 for more information.

Syntax

```
dsInt16_t  dsmInit  (dsUint32_t        *dsmHandle,
   dsmApiVersion *dsmApiVersionP,
   char          *clientNodeNameP,
   char          *clientOwnerNameP,
   char          *clientPasswordP,
   char          *applicationType,
   char          *configfile,
   char          *options);
```

Parameters

dsUint32_t *dsmHandle (O)
The handle that identifies this initialization session and associates it with subsequent Tivoli Storage Manager calls.

dsmApiVersion *dsmApiVersionP (I)
A pointer to the data structure identifying the version of the API that the application client is using for this session. The structure contains the values of the three constants, DSM_API_VERSION, DSM_API_RELEASE, and DSM_API_LEVEL, that are set in the dsmapitd.h file. A previous call to **dsmQueryApiVersion** must be performed to ensure that compatibility exists between the application client API version and the version of the API library that is installed on the user's workstation.

char *clientNodeNameP (I)
This parameter is a pointer to the node for the Tivoli Storage Manager session. All sessions must have a node name associated with them. The constant, DSM_MAX_NODE_LENGTH, in the dsmapitd.h file sets the maximum size that is permitted for a node name.

The node name is not case-sensitive.

If this parameter is set both to NULL and *passwordaccess* is set to *prompt*, the API attempts to obtain the node name first from the options string that was passed. If it is not there, the API then attempts to obtain the node name from the configuration file or options files. If these attempts to find the node name fail, the UNIX or Linux API uses the system host name, while APIs on other operating systems return the DSM_RC_REJECT_ID_UNKNOWN code.

This parameter must be NULL if the *passwordaccess* option in the dsm.sys file is set to *generate*. The API uses the system host name.

char *clientOwnerNameP (I)
This parameter is a pointer to the owner of the Tivoli Storage Manager session. If the operating system on which the session starts is a multi-user operating system, an owner name of NULL (the root user) has the authority to back up, archive, restore, or retrieve any objects belonging to the application, regardless of the owner of the object.

The owner name is case-sensitive.

This parameter must be NULL if the *passwordaccess* option in the dsm.sys file is set to *generate*. The API then uses the login user ID.

Note: On a multi-user operating system, if *passwordaccess* is set to *prompt*, it is not necessary for the owner name to match the active user ID of the session running the application.

char *clientPasswordP (I)
This parameter is a pointer to the password of the node on which the Tivoli Storage Manager session runs. The DSM_MAX_VERIFIER_LENGTH constant in the dsmapitd.h file sets the maximum size that is permitted for a password.

The password is not case-sensitive.

Except when the password file is first started, the value of this parameter is ignored if *passwordaccess* is set to *generate*.

char *applicationType (I)
This parameter identifies the application that is running the session. The application client defines the value.

Each time an API application client starts a session with the server, the application type (or platform) of the client is updated on the server. We recommend that the application type value contain an operating system abbreviation because this value is entered in the **platform** field on the server. The maximum string length is DSM_MAX_PLATFORM_LENGTH.

To see the current value of the application type, call **dsmQuerySessInfo**.

char *configfile (I)
This parameter points to a character string that contains the fully-qualified name of an API configuration file. Options specified in the API configuration file override their specification in the client options file. Options files are defined when Tivoli Storage Manager (client or API) is installed.

char *options (I)

Points to a character string that can contain user options such as:

- *Compressalways*
- *Servername* (UNIX or Linux only)
- *TCPServeraddr*
- *Fromnode*
- *Fromowner*
- *EnableClientEncryptKey*

The application client can use the option list to override the values of these options that the configuration file sets.

The format of the options is:

1. Each option that is specified in the option list begins with a dash (-) and is followed by the option keyword.
2. The keyword, in turn, is followed by an equal sign (=) and then followed by the option parameter.
3. If the option parameter contains a blank space, enclose the parameter with single or double quotes.
4. If more than one option is specified, separate the options with blanks.

If options are NULL, values for all options are taken from the user options file or the API configuration file.

Return codes

The return code numbers are provided in parentheses ( ).

Table 39. Return codes for dsmInit

Return code	Explanation
DSM_RC_ABORT_SYSTEM_ERROR (1)	The server has detected a system error and has notified the clients.
DSM_RC_REJECT_VERIFIER_EXPIRED (52)	Password has expired and must be updated.
DSM_RC_REJECT_ID_UNKNOWN (53)	Could not find the node name.
DSM_RC_AUTH_FAILURE (137)	There was an authentication failure.
DSM_RC_NO_STARTING_DELIMITER (148)	There is no starting delimiter in pattern.
DSM_RC_NEEDED_DIR_DELIMITER (149)	A directory delimiter is needed immediately before and after the “match directories” meta-string (“...”) and one was not located.
DSM_RC_NO_PASS_FILE (168)	The password file is not available.
DSM_RC_UNMATCHED_QUOTE (177)	An unmatched quote is in the option string.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	Unable to open the message text file.
DSM_RC_INVALID_OPT (400)	An entry in the option string is invalid.
DSM_RC_INVALID_DS_HANDLE (2014)	Invalid DSM handle.
DSM_RC_NO_OWNER_REQD (2032)	Owner parameter must be NULL when *passwordaccess* is set to *generate*.
DSM_RC_NO_NODE_REQD (2033)	Node parameter must be NULL when *passwordaccess* is set to *generate*.
DSM_RC_WRONG_VERSION (2064)	The API version for the application client has a higher value than the Tivoli Storage Manager version.
DSM_RC_PASSWD_TOOLONG (2103)	The password that was specified is too long.
DSM_RC_NO_OPT_FILE (2220)	A configuration file could not be located.

Table 39. Return codes for dsmInit (continued)

Return code	Explanation
DSM_RC_INVALID_KEYWORD (2221)	A keyword that was specified in an options string is invalid.
DSM_RC_PATTERN_TOO_COMPLEX (2222)	The include-exclude pattern is too complex for Tivoli Storage Manager to interpret.
DSM_RC_NO_CLOSING_BRACKET (2223)	There is no closing bracket in the pattern.
DSM_RC_INVALID_SERVER (2225)	For a multi-user environment, the server in the system configuration file was not found.
DSM_RC_NO_HOST_ADDR (2226)	Not enough information to connect to host.
DSM_RC_MACHINE_SAME (2227)	The nodename that is defined in the options file cannot be the same as the system host name.
DSM_RC_NO_API_CONFIGFILE (2228)	Cannot open the configuration file.
DSM_RC_NO_INCLEXCL_FILE (2229)	The include-exclude file was not found.
DSM_RC_NO_SYS_OR_INCLEXCL (2230)	Either the dsm.sys file or the include-exclude file was not found.

Related concepts:

Client options file overview

Processing options

dsmInitEx

The **dsmInitEx** function call starts an API session by using the additional parameters for extended verification.

Syntax

```
dsInt16_t  dsmInitEx   (dsUint32_t        *dsmHandleP,
                        dsmInitExIn_t     *dsmInitExInP,
                        dsmInitExOut_t    *dsmInitExOutP) ;
```

Parameters

dsUint32_t *dsmHandleP (O)
The handle that identifies this initialization session and associates it with subsequent Tivoli Storage Manager calls.

dsmInitExIn_t *dsmInitExInP
This structure contains the following input parameters:

dsmApiVersion *dsmApiVersionP (I)
This parameter is a pointer to the data structure that identifies the version of the API that the application client is using for this session. The structure contains the values of the four constants, DSM_API_VERSION, DSM_API_RELEASE, DSM_API_LEVEL, and DSM_API_SUBLEVEL that are set in the `dsmapitd.h` file. Call **dsmQueryApiVersionEx** and verify that the API version of the application client and the version of the API library that is installed on the user's workstation is compatible.

char *clientNodeNameP (I)
This parameter is a pointer to the node for the Tivoli Storage Manager session. All sessions must be associated with a node name. The DSM_MAX_NODE_LENGTH constant in the `dsmapitd.h` file sets the maximum size for a node name.

The node name is not case-sensitive.

If this parameter is set to NULL, and **passwordaccess** is set to prompt, the API attempts to obtain the node name first from the options string that was passed. If it is not there, the API then attempts to obtain the node name from the configuration file or options files. If these attempts to find the node name fail, the UNIX or Linux API uses the system host name, while the APIs from other operating systems return DSM_RC_REJECT_ID_UNKNOWN.

This parameter must be NULL if the **passwordaccess** option in the dsm.sys file is set to generate. The API then uses the system host name.

char *clientOwnerNameP (I)
This parameter is a pointer to the owner of the Tivoli Storage Manager session. If the operating system is a multi-user platform, an owner name of NULL (the root user) has the authority to back up, archive, restore, or retrieve any objects that belong to the application, regardless of the owner of the object.

The owner name is case-sensitive.

This parameter must be NULL if the **passwordaccess** option in the dsm.sys file is set to generate. The API then uses the login user ID.

Tip: On a multi-user platform, if **passwordaccess** is set to prompt, it is not necessary for the owner name to match the active user ID of the session that is running the application.

char *clientPasswordP (I)
A pointer to the password of the node on which the Tivoli Storage Manager session runs. The DSM_MAX_VERIFIER_LENGTH constant in the dsmapitd.h file sets the maximum size that is allowed for a password.

The password is not case-sensitive.

Except when the password file is first started, the value of this parameter is ignored if **passwordaccess** is set to generate.

char *userNameP;
A pointer to the administrative user name that has client authority for this node.

char *userPasswordP;
A pointer to the password for the **userName** parameter, if a value is supplied.

char *applicationType (I)
Identifies the application that is running the Tivoli Storage Manager session. The application client identifies the value.

Each time an API application client starts a session with the server, the application type (or operating system) of the client is updated on the server. The value is entered in the **platform** field on the server. Consider using an operating system ID in the value. The maximum string length is defined in the DSM_MAX_PLATFORM_LENGTH constant.

To view the current value of the application type, call **dsmQuerySessInfo**.

char *configfile (I)
Points to a character string that contains the fully qualified name of an API configuration file. Options that are specified in the API configuration file override their specification in the client options file. Options files are defined when Tivoli Storage Manager (client or API) is installed.

char *options (I)
Points to a character string that can contain user options such as:
- Compressalways
- Servername (UNIX and Linux systems only)
- TCPServeraddr (not for UNIX systems)
- Fromnode
- Fromowner

The application client can use the options list to override the values of these options that the configuration file sets.

Options have the following format:
1. Each option that is specified in the option list begins with a dash (-) and is followed by the option keyword.
2. The keyword is followed by an equal sign (=) and then the option parameter.
3. If the option parameter contains a blank space, enclose the parameter with single or double quotation marks.
4. If more than one option is specified, separate the options with blanks.

If options are NULL, the values for all options are taken from the user options file or the API configuration file.

dirDelimiter
The directory delimiter that is prefixed on the file space, high-level or low-level names. You must specify the **dirDelimiter** parameter only if the application overrides the system defaults. In a UNIX or Linux environment, the default is forward slash (/). In a Windows environment, the default is backslash (\).

useUnicode
A Boolean flag that indicates whether Unicode is enabled. The **useUnicode** flag must be false to achieve cross-platform interoperability between UNIX and Windows systems.

bCrossPlatform
A Boolean flag that must be set (bTrue) to achieve cross-platform interoperability between UNIX and Windows systems. When the bCrossPlatform flag is set, the API ensures that the file spaces are not Unicode and that the application does not use Unicode. A Windows application that uses Unicode is not compatible with applications that use non-Unicode encodings. The bCrossPlatform flag must not be set for a Windows application that uses Unicode.

UseTsmBuffers
Indicates whether to use buffer copy elimination.

numTsmBuffers
Number of buffers when useTsmBuffers=bTrue.

bEncryptKeyEnabled
Indicates whether encryption with application-managed key is used.

encryptionPasswordP
The encryption password.

Restriction: When encryptkey=save, if an encrypt key exists, the value that is specified in the **encryptionPasswordP** is ignored.

dsmAppVersion *appVersionP (I)
This parameter is a pointer to the data structure that identifies the version

information of the application that is starting an API session. The structure contains the values of the four constants, applicationVersion, applicationRelease, applicationLevel, and applicationSubLevel, which are set in the `tsmapitd.h` file.

dsmInitExOut_t *dsmInitExOut P
This structure contains the output parameters.

dsUint32_t *dsmHandle (0)
The handle that identifies this initialization session and associates it with subsequent API calls.

infoRC
Additional information about the return code. Check both the function return code and the value of **infoRC**. An **infoRC** value of DSM_RC_REJECT_LASTSESS_CANCELED (69), the Tivoli Storage Manager indicates that the administrator canceled the last session.

Return codes

The return code numbers are provided in parentheses ( ).

Table 40. Return codes for **dsmInitEx**

Return code	Explanation
DSM_RC_ABORT_SYSTEM_ERROR (1)	The Tivoli Storage Manager server detected a system error and notified the clients.
DSM_RC_REJECT_VERIFIER_EXPIRED (52)	Password expired and must be updated. The next call must be **dsmChangePW** with the handle returned on this call.
DSM_RC_REJECT_ID_UNKNOWN (53)	Cannot not find the node name.
DSM_RC_TA_COMM_DOWN (103)	The communications link is down.
DSM_RC_AUTH_FAILURE (137)	There was an authentication failure.
DSM_RC_NO_STARTING_DELIMITER (148)	There is no starting delimiter in pattern.
DSM_RC_NEEDED_DIR_DELIMITER (149)	A directory delimiter is needed immediately before and after the “match directories” meta-string (“...”), but was not found.
DSM_RC_NO_PASS_FILE (168)	The password file is not available.
DSM_RC_UNMATCHED_QUOTE (177)	An unmatched quotation mark is in the option string.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	Unable to open the message text file.
DSM_RC_INVALID_OPT (2013)	An entry in the option string is invalid.
DSM_RC_INVALID_DS_HANDLE (2014)	Invalid DSM handle.
DSM_RC_NO_OWNER_REQD (2032)	Owner parameter must be NULL when **passwordaccess** is set to `generate`.
DSM_RC_NO_NODE_REQD (2033)	Node parameter must be NULL when **passwordaccess** is set to generate.
DSM_RC_WRONG_VERSION (2064)	Application client's API version has a higher value than the Tivoli Storage Manager version.
DSM_RC_PASSWD_TOOLONG (2103)	The specified password is too long.
DSM_RC_NO_OPT_FILE (2220)	No configuration file is found.
DSM_RC_INVALID_KEYWORD (2221)	A keyword that is specified in an options string is invalid.
DSM_RC_PATTERN_TOO_COMPLEX (2222)	Include-exclude pattern too complex to be interpreted by Tivoli Storage Manager.
DSM_RC_NO_CLOSING_BRACKET (2223)	There is no closing bracket in the pattern.

Table 40. Return codes for **dsmInitEx** *(continued)*

Return code	Explanation
DSM_RC_INVALID_SERVER (2225)	For a multi-user environment, the server in the system configuration file was not found.
DSM_RC_NO_HOST_ADDR (2226)	Not enough information to connect to the host.
DSM_RC_MACHINE_SAME (2227)	The node name that is defined in the options file cannot be the same as the system host name.
DSM_RC_NO_API_CONFIGFILE (2228)	Cannot open the configuration file.
DSM_RC_NO_INCLEXCL_FILE (2229)	The include-exclude file was not found.
DSM_RC_NO_SYS_OR_INCLEXCL (2230)	Either the `dsm.sys` or the include-exclude file was not found.

Related concepts:

Client options file overview

Processing options

dsmLogEvent

The **dsmLogEvent** function call logs a user message (ANE4991 I) to the server log file, to the local error log, or to both. A structure of type **logInfo** is passed in the call. This call must be performed while at **InSession** state inside a session. Do not perform it within a send, get, or query. To retrieve messages logged on the server, use the **query actlog** command through the administrative client.

See the summary state diagram, Figure 20 on page 75.

Syntax

```
dsInt16_t dsmLogEvent
     (dsUint32_t    dsmHandle,
     logInfo        *logInfoP);
```

Parameters

dsUint32_t dsmHandle(I)
The handle that associates this call with a previous **dsmInitEx** call.

logInfo *logInfoP (I)
Passes the message and destination. The application client is responsible for allocating storage for the structure.

The fields in the **logInfo** structure are:

message
The text of the message to be logged. This must be a null-ended string. The maximum length is DSM_MAX_RC_MSG_LENGTH.

dsmLogtype
Specifies where to log the message. Possible values include: **logServer**, **logLocal**, **logBoth**.

Return codes

The return code numbers are provided in parentheses ( ).

Table 41. Return codes for dsmLogEvent

Return code	Explanation
DSM_RC_STRING_TOO_LONG (2120)	The message string is too long.

dsmLogEventEx

The **dsmLogEventEx** function call logs a user message to the server log file, to the local error log, or to both. This call must be made while at an **InSession** state within a session. The call cannot be made within a send, get, or query call.

Summary state diagram: For an overview of the session interactions, see the summary state diagram in the following topic:

Figure 20 on page 75

The severity determines the Tivoli Storage Manager message number. To view messages that are logged on the server, use the **query actlog** command through the administrative client. Use the Tivoli Storage Manager client option, `errorlogretention`, to prune the client error log file if the application generates numerous client messages written to the client log, `dsmLogType` either `logLocal` or `logBoth`. For more information, see the Tivoli Storage Manager server documentation.

Syntax

```
extern dsInt16_t DSMLINKAGE  dsmLogEventEx(
       dsUint32_t            dsmHandle,
       dsmLogExIn_t          *dsmLogExInP,
       dsmLogExOut_t         *dsmLogExOutP
);
```

Parameters

dsUint32_t dsmHandle(I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmLogExIn_t *dsmLogExInP
This structure contains the input parameters.

dsmLogSeverity severity;
This parameter is the event severity. The possible values are:

```
logSevInfo,       /* information ANE4990 */
logSevWarning,    /* warning     ANE4991 */
logSevError,      /* Error       ANE4992 */
logSevSevere      /* severe      ANE4993 */
```

char appMsgID[8];
This parameter is a string to identify the specific application message. A suitable format is three characters that are followed by four numbers, for example: `DSM0250`.

dsmLogType logType;
This parameter specifies where to direct the event. The parameter has the following possible values:

- `logServer`

- logLocal
- logBoth

char *message;
This parameter is the text of the event message to log. The text must be a null-ended string. The maximum length is DSM_MAX_RC_MSG_LENGTH.

Restriction: Messages that go to the server must be in English. Non-English messages do not display correctly.

dsmLogExOut_t *dsmLogExOutP
This structure contains the output parameters. Currently, there are no output parameters.

Return codes

The return code numbers are provided in parentheses ( ).

Table 42. Return codes for dsmLogEventEx

Return code	Explanation
DSM_RC_STRING_TOO_LONG (2120)	The message string is too long.

dsmQueryAccess

The **dsmQueryAccess** function call queries the server for all access authorization rules for either backup versions or archived copies of your objects. A pointer to an array of access rules is passed in to the call, and the completed array is returned. A pointer to the number of rules is passed in to indicate how many rules are in the array.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t DSMLINKAGE dsmQueryAccess
                  (dsUint32_t           dsmHandle),
                   qryRespAccessData    **accessListP,
                   dsUint16_t           *numberOfRules) ;
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

qryRespAccessData **accessListP (O)
A pointer to an array of qryRespAccessData elements that the API library allocates. Each element corresponds to an access rule. The number of elements in the array is returned in the **numberOfRules** parameter. The information that is returned in each qryRespAccessData element includes the following:

Name	Description
ruleNumber	The ID for the access rule. This identifies the rule for deletion.
AccessType	The backup or archive type.
Node	The node on which you gave access.
Owner	The user to whom you gave access.
objName	The high-level, or low-level file space descriptors.

dsUint32_t *numberOfRules (0)
Returns the number of rules in the accessList array.

dsmQueryApiVersion

The **dsmQueryApiVersion** function call performs a query request for the API library version that the application client accesses.

All updates to the API are made in an upward-compatible format. Any application client with an API version or release less than, or equal to, the API library on the end user's workstation operates without change. Be aware before you proceed that should the **dsmQueryApiVersion** call return a version or version release older than that of the application clients, some API calls might be enhanced in a manner that is not supported by the end user's older version of the API.

The application API version number is stored in the dsmapitd.h header file as constants DSM_API_VERSION, DSM_API_RELEASE, and DSM_API_LEVEL.

There are no return codes that are specific to this call.

Syntax

```
void dsmQueryApiVersion  (dsmApiVersion *apiVersionP);
```

Parameters

dsmApiVersion *apiVersionP (0)
This parameter is a pointer to the structure that contains the API library version, release, and level components. For example, if the library is version 1.1.0, then, after returning from the call, the fields of the structure contain the following values:

```
dsmApiVersionP->version  = 1
dsmApiVersionP->release  = 1
dsmApiVersionP->level    = 0
```

dsmQueryApiVersionEx

The **dsmQueryApiVersionEx** function call performs a query request for the API library version that the application client accesses.

All updates to the API are made in an upward-compatible format. Any application client that has an API version or release less than or equal to the API library on the end user's workstation operates without change. See Summary of Code Changes in the README_api_enu file for exceptions to upward compatibility. If the **dsmQueryApiVersionEx** call returns a version or version release that is different from that of the application client, be aware before you proceed that some API calls might be enhanced in a manner that is not supported by the end user's older version of the API.

The application API version number is stored in the dsmapitd.h header file as constants DSM_API_VERSION, DSM_API_RELEASE, DSM_API_LEVEL, and DSM_API_SUBLEVEL.

There are no return codes that are specific to this call.

Syntax

```
void dsmQueryApiVersionEx (dsmApiVersionEx *apiVersionP);
```

Parameters

dsmApiVersionEx *apiVersionP (O)

This parameter is a pointer to the structure that contains the API library's version, release, level, and sublevel components. For example, if the library is Version 5.5.0.0, then, after returning from the call, the fields of the structure contain the following values:

- `ApiVersionP->version  = 5`
- `ApiVersionP->release  = 5`
- `ApiVersionP->level    = 0`
- `ApiVersionP->subLevel = 0`

dsmQueryCliOptions

The **dsmQueryCliOptions** function call queries important option values in the user's option files. A structure of type **optStruct** is passed in the call and contains the information. This call is performed before **dsmInitEx** is called, and it determines the setup before the session.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmQueryCliOptions
    (optStruct     *optstructP);
```

Parameters

optStruct *optstructP (I/O)

This parameter passes the address of the structure that the API completes. The application client is responsible for allocating storage for the structure. On successful return, the appropriate information is entered in the fields in the structure.

The following information is returned in the **optStruct** structure:

Name	Description
dsmiDir	The value of the environment DSMI_DIR variable.
dsmiConfig	The client option file as specified by the DSMI_CONFIG environment variable.
serverName	The name of the Tivoli Storage Manager server.
commMethod	The communication method selected. See the #defines for DSM_COMM_* in the dsmapitd.h file.
serverAddress	The address of the server that is based on the communication method.
nodeName	The client node (machine) name.
compression	This field provides information regarding the compression option.
passwordAccess	The values are: *bTrue* for generate, and *bFalse* for prompt.

Related concepts:

Processing options

dsmQuerySessInfo

The **dsmQuerySessInfo** function call starts a query request to Tivoli Storage Manager for information related to the operation of the specified session in **dsmHandle**. A structure of type **ApiSessInfo** is passed in the call, with all available session related information entered. This call is started after a successful **dsmInitEx** call.

The information that is returned in the **ApiSessInfo** structure includes the following:

- Server information: port number, date and time, and type
- Client defaults: application type, delete permissions, delimiters, and transaction limits
- Session information: login ID, and owner
- Policy data: domain, active policy set, and retention grace period

See Appendix B, "API type definitions source files," on page 155 for information about the content of the structure that is passed and each field within it.

Syntax

```
dsInt16_t dsmQuerySessInfo  (dsUint32_t         dsmHandle,
   ApiSessInfo   *SessInfoP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

ApiSessInfo *SessInfoP (I/O)
This parameter passes the address of the structure that the API enters. The application client is responsible for allocating storage for the structure and for completing the field entries that indicate the version of the structure that is used. On successful return, the fields in the structure are completed with the appropriate information. The **adsmServerName** is the name that is given in the **define server** command on the Tivoli Storage Manager server. If the archiveRetentionProtection field is true, the server is enabled for retention protection.

Return codes

The return code numbers are provided in parentheses ( ).

Table 43. Return codes for dsmQuerySessInfo

Return code	Explanation
DSM_RC_NO_SESS_BLK (2006)	No server session block information.
DSM_RC_NO_POLICY_BLK (2007)	No server policy information available.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.

dsmQuerySessOptions

The **dsmQuerySessOptions** function call queries important option values that are valid in the specified session in **dsmHandle**. A structure of type optStruct is passed in the call and contains the information.

This call is started after a successful **dsmInitEx** call. The values that are returned might be different from the values returned on a **dsmQueryCliOptions** call, depending on values that are passed to the **dsmInitEx** call, primarily optString, and optFile. For information about option precedence, see "Understanding configuration and options files" on page 1.

There are no return codes that are specific to this call.

Syntax

```
dsInt16_t dsmQuerySessOptions
    (dsUint32_t    dsmHandle,
    optStruct      *optstructP);
```

Parameters

dsUint32_t dsmhandle(I)
The handle that associates this call with a previous **dsmInitEx** call.

optStruct *optstructP (I/O)
This parameter passes the address of the structure that the API completes. The application client is responsible for allocating storage for the structure. On successful return, the fields in the structure are completed with the appropriate information.

The information returned in the optStruct structure is:

Name	Description
dsmiDir	The value of the DSMI_DIR environment variable.
dsmiConfig	The dsm.opt file that the DSMI_CONFIG environment variable specifies.
serverName	The name of the Tivoli Storage Manager server stanza in the options file.
commMethod	The communication method that was selected. See the #defines for DSM_COMM_* in the dsmapitd.h file.
serverAddress	The address of the server that is based on the communication method.
nodeName	The name of the client's node (machine).
compression	The value of the compression option (bTrue=on and bFalse=off).
compressAlways	The value of the compressalways option (bTrue=on and bFalse=off).
passwordAccess	Value bTrue for generate, and bFalse for prompt.

Related concepts:

Processing options

dsmRCMsg

The **dsmRCMsg** function call obtains the message text that is associated with an API return code.

The **msg** parameter displays the message prefix return code in parentheses ( ), followed by the message text. For example, a call to **dsmRCMsg** might return the following:

```
ANS0264E (RC2300) Only root user can execute dsmChangePW or dsmDeleteFS.
```

For some languages where characters are different in ANSII and OEM code pages, it might be necessary to convert strings from ANSII to OEM before printing them out (for example, Eastern European single-byte character sets). The following is an example:

```
dsmRCMsg(dsmHangle, rc, msgBuf);
#ifdef WIN32
#ifndef WIN64
CharToOemBuff(msgBuf, msgBuf, strlen(msgBuf));
#endif
#endif
printf("
```

Syntax

```
dsInt16_t dsmRCMsg  (dsUint32_t       dsmHandle,
   dsInt16_t        dsmRC,
   char        *msg);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsInt16_t dsmRC (I)
The API return code of the associated message text. The API return codes are listed in the dsmrc.h file. See Appendix A, "API return codes source file: dsmrc.h," on page 145 for more information.

char *msg (O)
This parameter is the message text that is associated with the return code, **dsmRC**. The caller is responsible for allocating enough space for the message text.

The maximum length for **msg** is defined as DSM_MAX_RC_MSG_LENGTH.

On platforms that have National Language Support and a choice of language message files, the API returns a message string in the national language.

Return codes

The return code numbers are provided in parentheses ( ).

Table 44. Return codes for dsmRCMsg

Return code	Explanation
DSM_RC_NULL_MSG (2002)	The **msg** parameter for dsmRCMsg call is a NULL pointer.
DSM_RC_INVALID_RETCODE (2021)	Return code that was passed to **dsmRCMsg** call is an invalid code.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	Unable to open the message text file.

dsmRegisterFS

The **dsmRegisterFS** function call registers a new file space with the Tivoli Storage Manager server. Register a file space first before you can back up any data to it.

Application clients should not use the same file space names that a backup-archive client would use.

- On UNIX or Linux, run the **df** command for these names.
- On Windows, these names are generally the volume labels that are associated with the different drives on your system.

Syntax

```
dsInt16_t dsmRegisterFS  (dsUint32_t         dsmHandle,
   regFSData     *regFilespaceP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

regFSData *regFilespaceP (I)
This parameter passes the name of the file space and associated information that you need to register with the Tivoli Storage Manager server.

Tip: The *fstype* field includes the prefix, **"API:".** All file space queries display this string. For example, if the user passes *myfstype* for *fstype* in **dsmRegisterFS**, the actual value string on the server is returned as `API:myfstype` when queried. This prefix distinguishes API objects from backup-archive objects.

The usable area for **fsInfo** is now DSM_MAX_USER_FSINFO_LENGTH.

Return codes

The return code numbers are provided in parentheses ( ).

Table 45. Return codes for dsmRegisterFS

Return code	Explanation
DSM_RC_INVALID_FSNAME (2016)	Invalid file space name.
DSM_RC_INVALID_DRIVE_CHAR (2026)	Drive letter is not an alphabetic character.
DSM_RC_NULL_FSNAME (2027)	Null file space name.
DSM_RC_FS_ALREADY_REGED (2062)	File space is already registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_FSINFO_TOOLONG (2106)	File space information is too long.

dsmReleaseBuffer

The **dsmReleaseBuffer** function returns a buffer to Tivoli Storage Manager. The application calls **dsmReleaseBuffer** after a **dsmGetDataEx** was called and the application has moved all the data out of the buffer and is ready to release it. **dsmReleaseBuffer** requires that **dsmInitEx** was called with the ***UseTsmBuffers*** set to *btrue* and a non-zero value was provided for *numTsmBuffers*. **dsmReleaseBuffer** should also be called if the application is about to call **dsmTerminate** and it still holds IBM Tivoli Storage Manager buffers.

dsmReleaseBufferSyntax

```
dsInt16_t  dsmReleaseBuffer  (releaseBufferIn_t     *dsmReleaseBufferInP,
                              releaseBufferOut_t    *dsmReleaseBufferOutP) ;
```

Parameters

releaseBufferIn_t * dsmReleaseBufferInP (I)
This structure contains the following input parameters.

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsUint8_t tsmBufferHandle(I)
The handle that identifies this buffer.

char *dataPtr(I)
The address to which the application is written.

Return codes

The return code numbers are provided in parentheses ( ).

Table 46. Return codes for dsmReleaseBuffer

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE	The call was not issued in the proper state.
DSM_RC_INVALID_TSMBUFFER	The handle or the value of **dataPtr** are invalid.
DSM_RC_BUFF_ARRAY_ERROR	A buffer array error occurred.

dsmRenameObj

The **dsmRenameObj** function call renames the high-level or low-level object name. For backup objects, pass in the current object name and changes either for high-level or low-level object names. For archive objects, pass in the current object file space name and object ID, and changes either for high-level or low-level object names. Use this function call within **dsmBeginTxn** and **dsmEndTxn** calls.

The merge flag determines whether or not a duplicate backup object name is merged with the existing backups. If the new name corresponds to an existing object and merge is true, the current object is converted to the new name and it becomes the active version of the new name while the existing active object that had that name becomes the top most inactive copy of the object. If the new name corresponds to an existing object and merge is false, the function then returns the return code, DSM_RC_ABORT_DUPLICATE_OBJECT.

Restriction: Only the owner of the object can rename it.

The **dsmRenameObj** function call tests for these merge conditions:

- The current **dsmObjName** object and the new high-level or low-level object must match on owner, copy group, and management class.
- The current **dsmObjName** must have been backed up more recently than the currently active object with the new name.
- There must be only an active copy of the current **dsmObjName** with no inactive copies.

Syntax

```
dsInt16_t dsmRenameObj  (dsmRenameIn_t     *dsmRenameInP,
                         dsmRenameOut_t    *dsmRenameOutP);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmRenameIn_t *dsmRenameInP
This structure contains the input parameters.

dsUint8_t repository (I);
This parameter indicates whether the file space to delete is in the backup repository or the archive repository.

dsmObjName *objNameP (I);
This parameter is a pointer to the structure that contains the current file space name, high-level object name, low-level object name, and object type.

char newHl [DSM_MAX_HL_LENGTH + 1];
This parameter specifies the new high-level name.

char newLl [DSM_MAX_LL_LENGTH + 1];
This parameter specifies the new low-level name.

dsBool_t merge;
This parameter determines whether or not a backup object is merged with duplicate named objects. The values are either true or false.

ObjID;
The object ID for archive objects.

dsmRenameOut_t *dsmRnameOutP
This structure contains the output parameters.

Note: There are no output parameters.

Return codes

The return code numbers are provided in parentheses ( ).

Table 47. Return codes for dsmRenameObj

Return code	Explanation
DSM_RC_ABORT_MERGE_ERROR (45)	Server detected a merge error.
DSM_RC_ABORT_DUPLICATE_OBJECT (32)	Object already exists and merge is false.
DSM_RC_ABORT_NO_MATCH (2)	Object not found.
DSM_RC_REJECT_SERVER_DOWNLEVEL (58)	The Tivoli Storage Manager server must be at V3.7.4.0 or later for this function to work.

dsmRequestBuffer

The **dsmRequestBuffer** function returns a buffer to Tivoli Storage Manager. The application calls **dsmRequestBuffer** after a **dsmGetDataEx** was called and the application has moved all the data out of the buffer and is ready to release it.

dsmReleaseBuffer requires that **dsmInitEx** was called with the *UseTsmBuffers* set to *btrue* and a non-zero value was provided for *numTsmBuffers*. **dsmReleaseBuffer** should also be called if the application is about to call **dsmTerminate** and it still holds IBM Tivoli Storage Manager buffers.

Syntax

```
dsInt16_t  dsmRequestBuffer   (getBufferIn_t      *dsmRequestBufferInP,
                               getBufferOut_t     *dsmRequestBufferOutP) ;
```

Parameters

getBufferIn_t * dsmRequestBufferInP (I)
This structure contains the following input parameter:

dsUint32_t dsmHandle
The handle that identifies the session and associates it with a previous **dsmInitEx** call.

getBufferOut_t *dsmRequestBufferOut P (0)
This structure contains the output parameters.

dsUint8_t tsmBufferHandle(0)
The handle that identifies this buffer.

char *dataPtr(0)
The address to which application is written.

dsUint32_t *bufferLen(0)
Maximum number of bytes that can be written to this buffer.

Return codes

The return code numbers are provided in parentheses ( ).

Table 48. Return codes for dsmRequestBuffer

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE (33)	The call was not issued in the proper state.
DSM_RC_SENDDATA_WITH_ZERO_SIZE (34)	If the object being sent is 0 length, no calls to **dsmReleaseBuffer** are allowed.
DSM_RC_BUFF_ARRAY_ERROR (121)	A valid buffer could not be obtained.

dsmRetentionEvent

The **dsmRetentionEvent** function call sends a list of object IDs to the server, with a retention event operation to be performed on these objects. Use this function call within **dsmBeginTxn** and **dsmEndTxn** calls.

Note: The Tivoli Storage Manager server must be at V5.2.2.0 or later for this function to work.

The maximum number of objects in a call is limited to the value of *maxObjPerTxn* that is returned in the *ApisessInfo* structure from a **dsmQuerySessInfo** call.

Only an owner of an object can send an event on that object.

The following events are possible:

eventRetentionActivate
Can be issued only for objects that are bound to an event based management class. Sending this event activates the event for this object and the state of the retention for this object changes from DSM_ARCH_RETINIT_PENDING to DSM_ARCH_RETINIT_STARTED.

eventHoldObj
This event issues a retention or deletion hold on the object so that, until a release is issued, the object is not expired and cannot be deleted.

eventReleaseObj
This event can only be issued for an object that has a value of DSM_ARCH_HELD_TRUE in the ***objectHeld*** field and removes the hold on the object resuming the original retention policy.

Before you send ***dsmRetentionEvent***, send the query sequence that is described in "Querying the Tivoli Storage Manager system" on page 33 to obtain the information for the object. The call to **dsmGetNextQObj** returns a data structure named **qryRespArchiveData** for archive queries. This data structure contains the information that is needed for **dsmRetentionEvent**.

Syntax

```
extern dsInt16_t DSMLINKAGE dsmRetentionEvent(
  dsmRetentionEventIn_t          *ddsmRetentionEventInP,
  dsmRetentionEventOut_t         *dsmRetentionEventOutP
   );
```

Parameters

dsmRetentionEventIn_t *dsmRetentionEventP
This structure contains the following input parameters:

dsUint16_t stVersion;
This parameter indicates the structure version.

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmEventType_t evenType (I);
This parameter indicates the event type. See the beginning of this section for the meaning of these possible values: **eventRetentionActivate**, **eventHoldObj**, **eventReleaseObj**

dsmObjList_t objList;
This parameter indicates a list of object IDs to signal.

Return codes

The return code numbers are provided in parentheses ( ).

Table 49. Return codes for dsmRetentionEvent

Return code	Explanation
DSM_RC_ABORT_NODE_NOT_AUTHORIZED (36)	The node or user does not have proper authority.
DSM_RC_ABORT_TXN_LIMIT_EXCEEDED (249)	Too many objects in the transaction.
DSM_RC_ABORT_OBJECT_ALREADY_HELD (250)	Object is already held, cannot issue another hold.
DSM_RC_REJECT_SERVER_DOWNLEVEL (58)	The Tivoli Storage Manager server must be at V5.2.2.0 or later for this function to work.

dsmSendBufferData

The **dsmSendBufferData** function call sends a byte stream of data to Tivoli Storage Manager through a buffer that was provided in a previous **dsmReleaseBuffer** call. The application client can pass any type of data for storage on the server. Usually this data are file data, but it is not limited to file data. You can call **dsmSendBufferData** several times, if the byte stream of data that you are sending is large. Regardless of whether the call succeeds or fails, the buffer is released.

Restriction: When you use the *useTsmBuffers* option, even if an object is included for compression, the object is not compressed.

Syntax

```
dsInt16_t  dsmSendBufferData   (sendBufferDataIn_t        *dsmSendBufferDataExInP,
                                sendBufferDataOut_t       *dsmSendBufferDataOutP) ;
```

Parameters

sendBufferDataIn_t * dsmSendBufferDataInP (I)
This structure contains the following input parameters.

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsUint8_t tsmBufferHandle(I)
The handle that identifies the buffer to send.

char *dataPtr(I)
The address to which application data was written.

dsUint32_t numBytes(I)
The actual number of bytes written by the application (should always be less than the value provided in **dsmReleaseBuffer**).

Return codes

The return code numbers are provided in parentheses ( ).

Table 50. Return codes for dsmSendBufferData

Return code	Explanation
DSM_RC_BAD_CALL_SEQUENCE (2041)	The call was not issued in the proper state.
DSM_RC_INVALID_TSMBUFFER (2042)	The handle or the value of **dataPtr** are invalid.
DSM_RC_BUFF_ARRAY_ERROR (2045)	A buffer array error occurred.
DSM_RC_TOO_MANY_BYTES (2043)	The value of ***numBytes*** is bigger than the size of the buffer provided in the **dsmReleaseBuffer** call.

dsmSendData

The **dsmSendData** function call sends a byte stream of data to Tivoli Storage Manager through a buffer. The application client can pass any type of data for storage on the server. Usually, these data are file data, but are not limited to such. You can call **dsmSendData** several times, if the byte stream of data that you want to send is large.

Restriction: The application client cannot reuse the buffer that is specified in **dsmSendData** until the **dsmSendData** call returns.

Tip: If Tivoli Storage Manager returns code 157 (DSM_RC_WILL_ABORT), start a call to **dsmEndSendObj** and then to **dsmEndTxn** with a vote of DSM_VOTE_COMMIT. The application then receives return code 2302 (DSM_RC_CHECK_REASON_CODE) and passes the reason code back to the application user. This informs the user why the server is ending the transaction.

Syntax

```
dsInt16_t dsmSendData  (dsUint32_t   dsmHandle,
   DataBlk *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

DataBlk *dataBlkPtr (I/O)
This parameter points to a structure that includes both a pointer to the buffer from which the data are to be sent, as well as the size of the buffer. On return, this structure contains the number of bytes that is actually transferred. See Appendix B, "API type definitions source files," on page 155 for the type definition.

Return codes

The return code numbers are provided in parentheses ( ).

Table 51. Return codes for dsmSendData

Return code	Explanation
DSM_RC_NO_COMPRESS_MEMORY (154)	Insufficient memory available to perform data compression or expansion.

Table 51. Return codes for dsmSendData (continued)

Return code	Explanation
DSM_RC_COMPRESS_GREW (155)	During compression the compressed data grew in size compared to the original data.
DSM_RC_WILL_ABORT (157)	An unknown and unexpected error occurred, causing the transaction to halt.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different than the Tivoli Storage Manager library version.
DSM_RC_NEEDTO_ENDTXN (2070)	Need to end the transaction.
DSM_RC_OBJ_EXCLUDED (2080)	The include-exclude list excludes the object.
DSM_RC_OBJ_NOBCG (2081)	The object has no backup copy group and will not be sent to the server.
DSM_RC_OBJ_NOACG (2082)	The object has no archive copy group and is not sent to the server.
DSM_RC_SENDDATA_WITH_ZERO_SIZE (2107)	The object cannot send data with a zero byte *sizeEstimate*.

dsmSendObj

The **dsmSendObj** function call starts a request to send a single object to storage. Multiple **dsmSendObj** calls and associated **dsmSendData** calls can be made within the bounds of a transaction for performance reasons.

The **dsmSendObj** call processes the data for the object as a byte stream passed in memory buffers. The **dataBlkPtr** parameter in the **dsmSendObj** call permits the application client to either:

- Pass the data and the attributes (the attributes are passed through the **objAttrPtr**) of the object in a single call.
- Specify part of the object data through the **dsmSendObj** call and the remainder of the data through one or more **dsmSendData** calls.

Alternatively, the application client can specify only the attributes through the **dsmSendObj** call and specify the object data through one or more calls to **dsmSendData**. For this method, set **dataBlkPtr** to NULL on the **dsmSendObj** call.

Tip: For certain object types, byte stream data might not be associated with the data; for example, a directory entry with no extended attributes.

Before **dsmSendObj** is called, a preceding **dsmBindMC** call must be made to properly bind a management class to the object that you want to back up or archive. The API keeps this binding so that it can associate the proper management class with the object when it is sent to the server. If you permit the management class that is bound on a **dsmSendObj** call to default for an object type of directory (DSM_OBJ_DIRECTORY), the default might not be the default management class. Instead, the management class with the greatest retention time is used. If more than one management class exists with this retention time, the first one that is encountered is used.

Follow all object data that is sent to storage with a **dsmEndSendObj** call. If you do not have object data to send to the server, or all data was contained within the **dsmSendObj** call, start a **dsmEndSendObj** call before you can start another **dsmSendObj** call. If multiple data sends were required through the **dsmSendData** call, the **dsmEndSendObj** follows the last send to indicate the state change.

Tip: If Tivoli Storage Manager returns code 157 (DSM_RC_WILL_ABORT), start a call to **dsmEndTxn** with a vote of DSM_VOTE_COMMIT. The application receives return code 2302 (DSM_RC_CHECK_REASON_CODE) and passes the reason code back to the application user. This informs the user why the server is ending the transaction.

If the reason code is 11 (DSM_RS_ABORT_NO_REPOSIT_SPACE), it is possible that the *sizeEstimate* is too small for the actual amount of data. The application needs to determine a more accurate *sizeEstimate* and send the data again.

Syntax

```
dsInt16_t dsmSendObj  (dsUint32_t       dsmHandle,
   dsmSendType  sendType,
   void        *sendBuff,
   dsmObjName  *objNameP,
   ObjAttr     *objAttrPtr,
   DataBlk     *dataBlkPtr);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmSendType sendType (I)
This parameter specifies the type of send that is being performed. Possible values include:

Name	Description
stBackup	A backup object that is sent to the server.
stArchive	An archive object that is sent to the server.
stBackupMountWait	A backup object for which you want the server to wait until the necessary device, such as a tape, is mounted.
stArchiveMountWait	An archive object for which you want the server to wait until the necessary device, such as a tape, is mounted.

Note: Use the **MountWait** types if there is any possibility that your application user might send data to a tape.

void *sendBuff (I)
This parameter is a pointer to a structure that contains other information specific to the **sendType** on the call. Currently, only a **sendType** of **stArchive** has an associated structure. This structure is called **sndArchiveData** and it contains the archive description.

dsmObjName *objNameP (I)
This parameter is a pointer to the structure that contains the file space name, high-level object name, low-level object name, and object type. See "Object names and IDs" on page 23 for more information.

ObjAttr *objAttrPtr (I)
This parameter passes object attributes of interest to the application. See Appendix B, "API type definitions source files," on page 155 for the type definition.

The attributes are:

- **owner** refers to the owner of the object. Determining whether the owner is declared to be a specific name or an empty string is important when getting the object back from Tivoli Storage Manager storage. See "Accessing objects as session owner" on page 25 for more information.

- **sizeEstimate** is a best estimate of the total size of the data object to send to the server. Be as accurate as possible on this size, because the server uses this attribute for efficient space allocation and object placement within its storage resources.

  If the size estimate that you specified is significantly smaller than the actual number of bytes that are sent, the server might have difficulty allocating enough space and end the transaction with a reason code of 11 (DSM_RS_ABORT_NO_REPOSIT_SPACE).

  Note: The size estimate is for the total size of the data object in bytes.

  Objects with a size smaller than DSM_MIN_COMPRESS_SIZE do not compress.

  If your object has no bit data (only the attribute information from this call), the **sizeEstimate** should be zero.

  Note: Starting with Version 5.1.0, the copy destination within a transaction is not checked for consistency on zero-length objects.
- **objCompressed** is a Boolean value that states whether or not the object data have already been compressed.

  If the object is compressed (object *compressed=bTrue*), Tivoli Storage Manager does not try to compress it again. If it is not compressed, Tivoli Storage Manager decides whether to compress the object, based on the values of the `compression` option set by the administrator and set in the API configuration sources.

  If your application plans to use partial object restore or retrieve, you cannot compress the data while sending it. To enforce this, set *ObjAttr.objCompressed* to *bTrue*.
- **objInfo** saves information about the particular object.

  Restriction: Information is not stored here automatically. When this attribute is used, you must set the attribute, *objInfoLength*, to show the length of *objInfo*.
- **mcNameP** contains the name of a management class that overrides the management class that is obtained from **dsmBindMC**.
- **disableDeduplication** is a Boolean value. When it is set to true, this object is not deduplicated by the client.

DataBlk *dataBlkPtr (I/O)

This parameter points to a structure that includes both a pointer to the buffer of data that is to be backed up or archived and the size of that buffer. This parameter applies to **dsmSendObj** only. If you want to begin sending data on a subsequent **dsmSendData** call, rather than on the **dsmSendObj** call, set the buffer pointer in the DataBlk structure to `NULL`. On return, this structure contains the number of bytes that is actually transferred. See Appendix B, "API type definitions source files," on page 155 for the type definition.

Return codes

The return code numbers are provided in parentheses ( ).

Table 52. Return codes for dsmSendObj

Return code	Explanation
DSM_RC_NO_COMPRESS_MEMORY (154)	Insufficient memory available to perform data compression or expansion.

Table 52. Return codes for dsmSendObj (continued)

Return code	Explanation
DSM_RC_COMPRESS_GREW (155)	During compression, the compressed data grew in size compared to the original data.
DSM_RC_WILL_ABORT (157)	An unknown and unexpected error occurred, causing the transaction to be halted.
DSM_RC_TL_NOACG (186)	The management class for this file does not have a valid copy group for the send type.
DSM_RC_NULL_OBJNAME (2000)	Null object name.
DSM_RC_NULL_OBJATTRPTR (2004)	Null object attribute pointer.
DSM_RC_INVALID_OBJTYPE (2010)	Invalid object type.
DSM_RC_INVALID_OBJOWNER (2019)	Invalid object owner.
DSM_RC_INVALID_SENDTYPE (2022)	Invalid send type.
DSM_RC_WILDCHAR_NOTALLOWED (2050)	Wildcard characters not allowed.
DSM_RC_FS_NOT_REGISTERED (2061)	File space not registered.
DSM_RC_WRONG_VERSION_PARM (2065)	Application client's API version is different from the Tivoli Storage Manager library version.
DSM_RC_NEEDTO_ENDTXN (2070)	Need to end transaction.
DSM_RC_OBJ_EXCLUDED (2080)	The include-exclude list excluded the object.
DSM_RC_OBJ_NOBCG (2081)	The object has no backup copy group, and it is not sent to the server.
DSM_RC_OBJ_NOACG (2082)	The object has no archive copy group, and it is not sent to the server.
DSM_RC_DESC_TOOLONG (2100)	Description is too long.
DSM_RC_OBJINFO_TOOLONG (2101)	Object information is too long.
DSM_RC_HL_TOOLONG (2102)	High-level qualifier is too long.
DSM_RC_FILESPACE_TOOLONG (2104)	File space name is too long.
DSM_RC_LL_TOOLONG (2105)	Low-level qualifier is too long.
DSM_RC_NEEDTO_CALL_BINDMC (2301)	**dsmBindMC** must be called first.

dsmSetAccess

The **dsmSetAccess** function call gives other users or nodes access to backup versions or archived copies of your objects, access to all your objects, or access to a selective set. When you give access to another user, that user can query, restore, or retrieve your files. This command supports wildcards for the following fields: *fs*, *hl*, *ll*, *node*, *owner*.

Note: You cannot give access to both backup versions and archive copies by using a single command. You must specify either backup or archive.

Syntax

```
dsInt16_t DSMLINKAGE dsmSetAccess
        (dsUint32_t          dsmHandle,
         dsmSetAccessType    accessType,
         dsmObjName          *objNameP,
         char                *node,
         char                *owner);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmAccessType accessType (I)
This parameter specifies the type of objects for which you want to give access. Possible values include:

Name	Description
atBackup	Specifies that access is being set to backup objects.
atArchive	Specifies that the access is being set for archive objects.

dsmObjName *objNameP (I)
This parameter is a pointer to the structure that contains the file space name, the high-level object name, and the low-level object name.

Note: To specify all file spaces, use an asterisk (*) for the file space name.

char *node (I)
This parameter is a pointer to the node name for which access is given. For any node, specify an asterisk (*).

char *owner (I)
This parameter is a pointer to the user name on the node to which you gave access. For all users, specify an asterisk (*).

Return codes

The return code numbers are provided in parentheses ( ).

Table 53. Return codes for dsmSetAccess

Return code	Explanation
DSM_RC_INVALID_ACCESS_TYPE (2110)	Invalid access type specified.
DSM_RC_FILE_SPACE_NOT_FOUND (124)	Specified file space was not found on the server.
DSM_RC_QUERY_COMM_FAILURE (2111)	Communication error during server query.
DSM_RC_NO_FILES_BACKUP (2112)	No files were backed up for this file space.
DSM_RC_NO_FILES_ARCHIVE (2113)	No files were archived for this file space.
DSM_RC_INVALID_SETACCESS (2114)	Invalid formulation of set access.

dsmSetUp

The **dsmSetUp** function call overwrites environment variable values. Call **dsmSetUp** before **dsmInitEx**. The values that were passed in the **envSetUp** structure overwrite any existing environment variables or defaults. If you specify NULL for a field, values are taken from the environment. If you do not set a value, the values are taken from the defaults.

Requirements:

1. If you use **dsmSetUp**, always call **dsmTerminate** before **dsmCleanUp**.
2. API instrumentation can only be activated if the testflag INSTRUMENT: API is set in the configuration file and the **dsmSetUp** or **dsmCleanUp** calls are used in the application.

Syntax

```
dsInt16_t DSMLINKAGE dsmSetUp
        (dsBool_t    mtFlag,
         envSetUp    *envSetUpP);
```

Parameters

dsBool_t mtFlag (I)
This parameter specifies if the API will be used in a single thread, or a multithread mode. Values include:

```
DSM_SINGLETHREAD
DSM_MULTITHREAD
```

Requirement: The multithread flag must be on for LAN-free data transfer to occur.

envSetUp *envSetUpP(I)
This parameter is a pointer to the structure that holds the overwrite values. Specify NULL if you do not want to override existing environment variables. The fields in the **envSetUp** structure include:

Name	Description
dsmiDir	A fully-qualified directory path that contains a message file on UNIX or Linux. It also specifies the dsmtca and the dsm.sys directories.
dsmiConfig	The fully-qualified name of the client options file.
dsmiLog	The fully-qualified path of the error log directory.
argv	Pass the argv[0] name of the calling program if the application must run with authorized user authority. See "Setting the passwordaccess option to generate without TCA" on page 21 for more information.
logName	The file name for an error log if the application does not use dsierror.log.
inclExclCaseSensitive	Indicates whether include/exclude rules are case-sensitive or case-insensitive. This parameter can be used on Windows only, it is ignored elsewhere.

Return codes

The return code numbers are provided in parentheses ( ).

Table 54. Return codes for dsmSetUp

Return code	Explanation
DSM_RC_ACCESS_DENIED (106)	Access to the specified file or directory is denied.
DSM_RC_INVALID_OPT (0400)	An invalid option was found.
DSM_RC_NO_HOST_ADDR (0405)	The TCPSERVERADDRESS for this server is not defined in the server name stanza in the system options file.
DSM_RC_NO_OPT_FILE (0406)	The options file specified by filename cannot be found.
DSM_RC_MACHINE_SAME (0408)	The NODENAME defined in the options file cannot be the same as the system *HostName*.
DSM_RC_INVALID_SERVER (0409)	The system options file does not contain the SERVERNAME option.
DSM_RC_INVALID_KEYWORD (0410)	An invalid option keyword was found in the **dsmInitEx** configuration file, the option string, dsm.sys, or dsm.opt.
DSM_RC_PATTERN_TOO_COMPLEX (0411)	The include or exclude pattern issued is too complex to be accurately interpreted by Tivoli Storage Manager.

Table 54. Return codes for dsmSetUp (continued)

Return code	Explanation
DSM_RC_NO_CLOSING_BRACKET (0412)	The include or exclude pattern is incorrectly constructed. The closing bracket is missing.
DSM_RC_NLS_CANT_OPEN_TXT (0610)	The system is unable to open the message text file.
DSM_RC_NLS_INVALID_CNTL_REC (0612)	The system is unable to use the message text file.
DSM_RC_NOT_ADSM_AUTHORIZED (0927)	You must be the authorized user to have multithreading and *passwordaccess* generate.
DSM_RC_NO_INCLEXCL_FILE (2229)	The include-exclude file was not found.
DSM_RC_NO_SYS_OR_INCLEXCL (2230)	Either the dsm.sys or the include-exclude file was not found.

dsmTerminate

The **dsmTerminate** function call ends a session with the Tivoli Storage Manager server and cleans up the Tivoli Storage Manager environment.

Syntax

There are no return codes that are specific for this call.

```
dsInt16_t dsmTerminate (dsUint32_t dsmHandle);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmUpdateFS

The **dsmUpdateFS** function call updates a file space in Tivoli Storage Manager storage. This update ensures that the administrator has a current record of your file space.

Syntax

```
dsInt16_t dsmUpdateFS  (dsUint32_t      dsmHandle,
   char         *fs,
   dsmFSUpd     *fsUpdP,
   dsUint32_t    fsUpdAct);
```

Parameters

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

char *fs (I)
This parameter is a pointer to the file space name.

dsmFSUpd *fsUpdP (I)
This parameter is a pointer to the structure that has the correct fields for the update that you want. Complete only those fields that need updating.

dsUint32_t fsUpdAct (I)
A 2-byte bit map that indicates which of the fields to update. The bit masks have the following values:
- DSM_FSUPD_FSTYPE
- DSM_FSUPD_FSINFO

Tip: For Windows operating systems, the drive letter value from **dsmDOSAttrib** is also updated when **FSINFO** is selected.

- DSM_FSUPD_OCCUPANCY
- DSM_FSUPD_CAPACITY
- DSM_FSUPD_BACKSTARTDATE
- DSM_FSUPD_BACKCOMPLETEDATE

For a description of these bit masks, see the DSM_FSUPD definitions in the following topic: Appendix B, "API type definitions source files," on page 155.

Return codes

The following table lists return codes for the **dsmUpdateFS** function call.

Table 55. Return codes for dsmUpdateFS

Return code	Return code number	Description
DSM_RC_FS_NOT_REGISTERED	2061	File space name is not registered.
DSM_RC_WRONG_VERSION_PARM	2065	The API version of the application client is different from the Tivoli Storage Manager library version.
DSM_RC_FSINFO_TOOLONG	2106	File space information is too long.

dsmUpdateObj

The **dsmUpdateObj** function call updates the meta information associated with an active backup or archive object already on the server. The application bit data is not affected. To update an object, you must give a specific non-wildcard name. To update an archived object, set the **dsmSendType** to **stArchive**. Only the latest named archive object is updated.

You can only start the **dsmUpdateObj** call in the session state; it cannot be called inside a transaction because it performs its own transaction. And, you can update only one object at a time.

Restriction: On a UNIX or Linux operating system, if you change the owner field, you cannot query or restore the object unless you are the root user.

Syntax

```
dsInt16_t  dsmUpdateObj
   (dsUint32_t      dsmHandle,
    dsmSendType     sendType,
    void           *sendBuff,
    dsmObjName     *objNameP,
    ObjAttr        *objAttrPtr,  /* objInfo */
    dsUint16_t      objUpdAct);  /* action bit vector */
```

Parameters

The field descriptions are the same as those in **dsmSendObj**, with the following exceptions:

dsmObjName *objNameP (I)
You cannot use a wildcard.

ObjAttr *objAttrPtr (I)
The **objCompressed** field is ignored for this call.

Other differences are:

- **owner**. If you specify a new **owner** field, the owner changes.
- **sizeEstimate**. If you specify a non-zero value it should be the actual amount of data sent, in bytes. The value is stored in the Tivoli Storage Manager metadata for future use.
- **objInfo**. This attribute contains the new information to be placed in the **objInfo** field. Set the **objInfoLength** to the length of the new **obiInfo**.

dsUint16_t objUpdAct
The bit masks and possible actions for **objUpdAct** are:

DSM_BACKUPD_MC
Updates the management class for the object.

DSM_BACKUPD_OBJINFO
Updates **objInfo**, **objInfoLength**, and **sizeEstimate**.

DSM_BACKUPD_OWNER
Updates the owner of the object.

DSM_ARCHUPD_DESCR
Updates the **Description** field. Enter the value for the new description through the **SendBuff** parameter. See the sample program for proper use.

DSM_ARCHUPD_OBJINFO
Updates **objInfo**, **objInfoLength**, and **sizeEstimate**.

DSM_ARCHUPD_OWNER
Updates the owner of the object.

Return codes

The return code numbers are provided in parentheses ( ).

Table 56. Return codes for dsmUpdateObj

Return code	Explanation
DSM_RC_INVALID_ACTION (2232)	Invalid action.
DSM_RC_FS_NOT_REGISTERED (2061)	File space not registered.
DSM_RC_BAD_CALL_SEQUENCE (2041)	Sequence of calls is invalid.
DSM_RC_WILDCHAR_NOTALLOWED (2050)	Wildcard characters are not allowed.
DSM_RC_ABORT_NO_MATCH (2)	Previous query does not match.

dsmUpdateObjEx

The **dsmUpdateObjEx** function call updates the meta information that is associated with an active backup or archive object that is on the server. The application bit data is not affected. To update an object, you must specify a non-wildcard name, or you can specify the object ID to update a specific archived object. You cannot use wildcard characters when specifying the name. To update a backup object, set the **dsmSendType** parameter to **stBackup**. To update an archived object, set the **dsmSendType** parameter to **stArchive**.

You can only start the **dsmUpdateObjEx** call in the session state; it cannot be called inside a transaction because it performs its own transaction. You can update only one object at a time.

Restriction: On a UNIX or Linux operating system, if you change the owner field, you cannot query or restore the object unless you are the root user. Only the current active version of a backup object can be updated.

Syntax

```
dsInt16_t  dsmUpdateObjEx
   (dsmUpdateObjExIn_t   *dsmUpdateObjExInP,
    dsmUpdateObjExOut_t  *dsmUpdateObjExOutP);
```

Parameters

dsmUpdateObjExIn_t *dsmUpdateObjExInP
This structure contains the following input parameters:

dsUint16_t stVersion (I)
The current version of the structure that is used.

dsUint32_t dsmHandle (I)
The handle that associates this call with a previous **dsmInitEx** call.

dsmSendType sendType (I)
The type of send that is being performed. The value can be:

stBackup
A backup object that is sent to the server.

stArchive
An archive object that is sent to the server.

dsmObjName *objNameP (I)
A pointer to the structure that contains the filespace name, high-level object name, low-level object name, and object type. You cannot use a wildcard.

ObjAttr *objAttrPtr (I)
Passes object attributes to the application. The values that are updated depend on the flags in the **objUpdAct** field. The **objCompressed** attribute is ignored for this call.

The attributes are:

- **owner** changes the owner if a new name is entered.
- **sizeEstimate** is the actual amount of data that is sent in bytes. The value is stored in the Tivoli Storage Manager meta data for future use.
- **objCompressed** is a Boolean value that states whether or not the object data have already been compressed.
- **objInfo** is an attribute that contains the new information to be placed in the **objInfo** field. Set the **objInfoLength** to the length of the new **objInfo**.
- **mcNameP** contains the name of a management class that overrides the management class that is obtained from **dsmBindMC**.

dsUint32_t objUpdAct
Specifies the bit masks and actions for **objUpdAct** are:

DSM_BACKUPD_MC
Updates the management class for the object.

DSM_BACKUPD_OBJINFO
Updates the information object (**objInfo**), the length of the information object (**objInfoLength**), and the amount of data that is sent (**sizeEstimate**) for the backup object.

DSM_BACKUPD_OWNER
Updates the owner for the backup object.

DSM_ARCHUPD_DESCR
Updates the **Description** field for the archive object. Enter the value for the new description through the **sendBuff** parameter.

DSM_ARCHUPD_OBJINFO
Updates the information object (**objInfo**), the length of the information object (**objInfoLength**), and the amount of data that is sent (**sizeEstimate**) for the archive object.

DSM_ARCHUPD_OWNER
Updates the owner of the archive object.

ObjID archObjId
Specifies the unique object ID for a specific archive object. Because multiple archive objects can have the same name, this parameter identifies a specific one. You can obtain the object ID by using a query archive call.

dsmUpdateObjExOut_t *dsmUpdateObjExOutP
This structure contains the output parameter:

dsUint16_t stVersion (I)
The current version of the structure that is used.

Return codes

The return code numbers are provided in parentheses ( ) in the following table.

Table 57. Return codes for dsmUpdateObjEx

Return code	Explanation
DSM_RC_INVALID_ACTION (2012)	Invalid action.
DSM_RC_FS_NOT_REGISTERED (2061)	File space not registered.
DSM_RC_BAD_CALL_SEQUENCE (2041)	Sequence of calls is invalid.
DSM_RC_WILDCHAR_NOTALLOWED (2050)	Wildcard characters are not allowed.
DSM_RC_ABORT_NO_MATCH (2)	Previous query does not match.

Appendix A. API return codes source file: dsmrc.h

The dsmrc.h header file contains all return codes that the API can return to an application.

The information that is provided here contains a point-in-time copy of the dsmrc.h file that is distributed with the API. View the file in the API distribution package for the latest version.

```
/*********************************************************************
* Tivoli Storage Manager                                             *
* API Client Component                                               *
*                                                                    *
* (C) Copyright IBM Corporation 1993,2010                            *
*********************************************************************/

/*********************************************************************/
/* Header File Name:  dsmrc.h                                        */
/*                                                                   */
/* Descriptive-name:  Return codes from Tivoli Storage Manager APIs  */
/*********************************************************************/
#ifndef _H_DSMRC
#define _H_DSMRC


#ifndef DSMAPILIB

#ifndef _H_ANSMACH
typedef int RetCode ;
#endif

#endif


#define DSM_RC_SUCCESSFUL               0 /* successful completion   */
#define DSM_RC_OK                       0 /* successful completion   */

#define DSM_RC_UNSUCCESSFUL            -1 /* unsuccessful completion */

/* dsmEndTxn reason code */
#define DSM_RS_ABORT_SYSTEM_ERROR          1
#define DSM_RS_ABORT_NO_MATCH              2
#define DSM_RS_ABORT_BY_CLIENT             3
#define DSM_RS_ABORT_ACTIVE_NOT_FOUND      4
#define DSM_RS_ABORT_NO_DATA               5
#define DSM_RS_ABORT_BAD_VERIFIER          6
#define DSM_RS_ABORT_NODE_IN_USE           7
#define DSM_RS_ABORT_EXPDATE_TOO_LOW       8
#define DSM_RS_ABORT_DATA_OFFLINE          9
#define DSM_RS_ABORT_EXCLUDED_BY_SIZE      10
#define DSM_RS_ABORT_NO_STO_SPACE_SKIP     11
#define DSM_RS_ABORT_NO_REPOSIT_SPACE      DSM_RS_ABORT_NO_STO_SPACE_SKIP
#define DSM_RS_ABORT_MOUNT_NOT_POSSIBLE    12
#define DSM_RS_ABORT_SIZESTIMATE_EXCEED    13
#define DSM_RS_ABORT_DATA_UNAVAILABLE      14
#define DSM_RS_ABORT_RETRY                 15
#define DSM_RS_ABORT_NO_LOG_SPACE          16
#define DSM_RS_ABORT_NO_DB_SPACE           17
#define DSM_RS_ABORT_NO_MEMORY             18

#define DSM_RS_ABORT_FS_NOT_DEFINED        20
#define DSM_RS_ABORT_NODE_ALREADY_DEFED    21
#define DSM_RS_ABORT_NO_DEFAULT_DOMAIN     22
#define DSM_RS_ABORT_INVALID_NODENAME      23
#define DSM_RS_ABORT_INVALID_POL_BIND      24
#define DSM_RS_ABORT_DEST_NOT_DEFINED      25
#define DSM_RS_ABORT_WAIT_FOR_SPACE        26
#define DSM_RS_ABORT_NOT_AUTHORIZED        27
#define DSM_RS_ABORT_RULE_ALREADY_DEFED    28
#define DSM_RS_ABORT_NO_STOR_SPACE_STOP    29

#define DSM_RS_ABORT_LICENSE_VIOLATION     30
```

© Copyright IBM Corp. 1993, 2016

```
#define DSM_RS_ABORT_EXTOBJID_ALREADY_EXISTS 31
#define DSM_RS_ABORT_DUPLICATE_OBJECT        32

#define DSM_RS_ABORT_INVALID_OFFSET          33    /* Partial Object Retrieve */
#define DSM_RS_ABORT_INVALID_LENGTH          34    /* Partial Object Retrieve */
#define DSM_RS_ABORT_STRING_ERROR            35
#define DSM_RS_ABORT_NODE_NOT_AUTHORIZED     36
#define DSM_RS_ABORT_RESTART_NOT_POSSIBLE    37
#define DSM_RS_ABORT_RESTORE_IN_PROGRESS     38
#define DSM_RS_ABORT_SYNTAX_ERROR            39

#define DSM_RS_ABORT_DATA_SKIPPED            40
#define DSM_RS_ABORT_EXCEED_MAX_MP           41
#define DSM_RS_ABORT_NO_OBJSET_MATCH         42
#define DSM_RS_ABORT_PVR_ERROR               43
#define DSM_RS_ABORT_BAD_RECOGTOKEN          44
#define DSM_RS_ABORT_MERGE_ERROR             45
#define DSM_RS_ABORT_FSRENAME_ERROR          46
#define DSM_RS_ABORT_INVALID_OPERATION       47
#define DSM_RS_ABORT_STGPOOL_UNDEFINED       48
#define DSM_RS_ABORT_INVALID_DATA_FORMAT     49
#define DSM_RS_ABORT_DATAMOVER_UNDEFINED     50

#define DSM_RS_ABORT_INVALID_MOVER_TYPE      231
#define DSM_RS_ABORT_ITEM_IN_USE             232
#define DSM_RS_ABORT_LOCK_CONFLICT           233
#define DSM_RS_ABORT_SRV_PLUGIN_COMM_ERROR   234
#define DSM_RS_ABORT_SRV_PLUGIN_OS_ERROR     235
#define DSM_RS_ABORT_CRC_FAILED              236
#define DSM_RS_ABORT_INVALID_GROUP_ACTION    237
#define DSM_RS_ABORT_DISK_UNDEFINED          238
#define DSM_RS_ABORT_BAD_DESTINATION         239
#define DSM_RS_ABORT_DATAMOVER_NOT_AVAILABLE 240
#define DSM_RS_ABORT_STGPOOL_COPY_CONT_NO    241
#define DSM_RS_ABORT_RETRY_SINGLE_TXN        242
#define DSM_RS_ABORT_TOC_CREATION_FAIL       243
#define DSM_RS_ABORT_TOC_LOAD_FAIL           244
#define DSM_RS_ABORT_PATH_RESTRICTED         245
#define DSM_RS_ABORT_NO_LANFREE_SCRATCH      246
#define DSM_RS_ABORT_INSERT_NOT_ALLOWED      247
#define DSM_RS_ABORT_DELETE_NOT_ALLOWED      248
#define DSM_RS_ABORT_TXN_LIMIT_EXCEEDED      249
#define DSM_RS_ABORT_OBJECT_ALREADY_HELD     250
#define DSM_RS_ABORT_INVALID_CHUNK_REFERENCE 254
#define DSM_RS_ABORT_DESTINATION_NOT_DEDUP   255
#define DSM_RS_ABORT_DESTINATION_POOL_CHANGED 257
#define DSM_RS_ABORT_NOT_ROOT                258


/* RETURN CODE */

#define DSM_RC_ABORT_SYSTEM_ERROR            DSM_RS_ABORT_SYSTEM_ERROR
#define DSM_RC_ABORT_NO_MATCH                DSM_RS_ABORT_NO_MATCH
#define DSM_RC_ABORT_BY_CLIENT               DSM_RS_ABORT_BY_CLIENT
#define DSM_RC_ABORT_ACTIVE_NOT_FOUND        DSM_RS_ABORT_ACTIVE_NOT_FOUND
#define DSM_RC_ABORT_NO_DATA                 DSM_RS_ABORT_NO_DATA
#define DSM_RC_ABORT_BAD_VERIFIER            DSM_RS_ABORT_BAD_VERIFIER
#define DSM_RC_ABORT_NODE_IN_USE             DSM_RS_ABORT_NODE_IN_USE
#define DSM_RC_ABORT_EXPDATE_TOO_LOW         DSM_RS_ABORT_EXPDATE_TOO_LOW
#define DSM_RC_ABORT_DATA_OFFLINE            DSM_RS_ABORT_DATA_OFFLINE
#define DSM_RC_ABORT_EXCLUDED_BY_SIZE        DSM_RS_ABORT_EXCLUDED_BY_SIZE

#define DSM_RC_ABORT_NO_REPOSIT_SPACE        DSM_RS_ABORT_NO_STO_SPACE_SKIP
#define DSM_RC_ABORT_NO_STO_SPACE_SKIP       DSM_RS_ABORT_NO_STO_SPACE_SKIP

#define DSM_RC_ABORT_MOUNT_NOT_POSSIBLE      DSM_RS_ABORT_MOUNT_NOT_POSSIBLE
#define DSM_RC_ABORT_SIZESTIMATE_EXCEED      DSM_RS_ABORT_SIZESTIMATE_EXCEED
#define DSM_RC_ABORT_DATA_UNAVAILABLE        DSM_RS_ABORT_DATA_UNAVAILABLE
#define DSM_RC_ABORT_RETRY                   DSM_RS_ABORT_RETRY
#define DSM_RC_ABORT_NO_LOG_SPACE            DSM_RS_ABORT_NO_LOG_SPACE
#define DSM_RC_ABORT_NO_DB_SPACE             DSM_RS_ABORT_NO_DB_SPACE
#define DSM_RC_ABORT_NO_MEMORY               DSM_RS_ABORT_NO_MEMORY

#define DSM_RC_ABORT_FS_NOT_DEFINED          DSM_RS_ABORT_FS_NOT_DEFINED
#define DSM_RC_ABORT_NODE_ALREADY_DEFED      DSM_RS_ABORT_NODE_ALREADY_DEFED
#define DSM_RC_ABORT_NO_DEFAULT_DOMAIN       DSM_RS_ABORT_NO_DEFAULT_DOMAIN
#define DSM_RC_ABORT_INVALID_NODENAME        DSM_RS_ABORT_INVALID_NODENAME
#define DSM_RC_ABORT_INVALID_POL_BIND        DSM_RS_ABORT_INVALID_POL_BIND
#define DSM_RC_ABORT_DEST_NOT_DEFINED        DSM_RS_ABORT_DEST_NOT_DEFINED
```

```
#define DSM_RC_ABORT_WAIT_FOR_SPACE          DSM_RS_ABORT_WAIT_FOR_SPACE
#define DSM_RC_ABORT_NOT_AUTHORIZED          DSM_RS_ABORT_NOT_AUTHORIZED
#define DSM_RC_ABORT_RULE_ALREADY_DEFED      DSM_RS_ABORT_RULE_ALREADY_DEFED
#define DSM_RC_ABORT_NO_STOR_SPACE_STOP      DSM_RS_ABORT_NO_STOR_SPACE_STOP

#define DSM_RC_ABORT_LICENSE_VIOLATION       DSM_RS_ABORT_LICENSE_VIOLATION
#define DSM_RC_ABORT_EXTOBJID_ALREADY_EXISTS DSM_RS_ABORT_EXTOBJID_ALREADY_EXISTS
#define DSM_RC_ABORT_DUPLICATE_OBJECT        DSM_RS_ABORT_DUPLICATE_OBJECT

#define DSM_RC_ABORT_INVALID_OFFSET          DSM_RS_ABORT_INVALID_OFFSET
#define DSM_RC_ABORT_INVALID_LENGTH          DSM_RS_ABORT_INVALID_LENGTH

#define DSM_RC_ABORT_STRING_ERROR            DSM_RS_ABORT_STRING_ERROR
#define DSM_RC_ABORT_NODE_NOT_AUTHORIZED     DSM_RS_ABORT_NODE_NOT_AUTHORIZED
#define DSM_RC_ABORT_RESTART_NOT_POSSIBLE    DSM_RS_ABORT_RESTART_NOT_POSSIBLE
#define DSM_RC_ABORT_RESTORE_IN_PROGRESS     DSM_RS_ABORT_RESTORE_IN_PROGRESS
#define DSM_RC_ABORT_SYNTAX_ERROR            DSM_RS_ABORT_SYNTAX_ERROR

#define DSM_RC_ABORT_DATA_SKIPPED            DSM_RS_ABORT_DATA_SKIPPED
#define DSM_RC_ABORT_EXCEED_MAX_MP           DSM_RS_ABORT_EXCEED_MAX_MP
#define DSM_RC_ABORT_NO_OBJSET_MATCH         DSM_RS_ABORT_NO_OBJSET_MATCH
#define DSM_RC_ABORT_PVR_ERROR               DSM_RS_ABORT_PVR_ERROR
#define DSM_RC_ABORT_BAD_RECOGTOKEN          DSM_RS_ABORT_BAD_RECOGTOKEN
#define DSM_RC_ABORT_MERGE_ERROR             DSM_RS_ABORT_MERGE_ERROR
#define DSM_RC_ABORT_FSRENAME_ERROR          DSM_RS_ABORT_FSRENAME_ERROR
#define DSM_RC_ABORT_INVALID_OPERATION       DSM_RS_ABORT_INVALID_OPERATION
#define DSM_RC_ABORT_STGPOOL_UNDEFINED       DSM_RS_ABORT_STGPOOL_UNDEFINED
#define DSM_RC_ABORT_INVALID_DATA_FORMAT     DSM_RS_ABORT_INVALID_DATA_FORMAT
#define DSM_RC_ABORT_DATAMOVER_UNDEFINED     DSM_RS_ABORT_DATAMOVER_UNDEFINED

#define DSM_RC_ABORT_INVALID_MOVER_TYPE      DSM_RS_ABORT_INVALID_MOVER_TYPE
#define DSM_RC_ABORT_ITEM_IN_USE             DSM_RS_ABORT_ITEM_IN_USE
#define DSM_RC_ABORT_LOCK_CONFLICT           DSM_RS_ABORT_LOCK_CONFLICT
#define DSM_RC_ABORT_SRV_PLUGIN_COMM_ERROR   DSM_RS_ABORT_SRV_PLUGIN_COMM_ERROR
#define DSM_RC_ABORT_SRV_PLUGIN_OS_ERROR     DSM_RS_ABORT_SRV_PLUGIN_OS_ERROR
#define DSM_RC_ABORT_CRC_FAILED              DSM_RS_ABORT_CRC_FAILED
#define DSM_RC_ABORT_INVALID_GROUP_ACTION    DSM_RS_ABORT_INVALID_GROUP_ACTION
#define DSM_RC_ABORT_DISK_UNDEFINED          DSM_RS_ABORT_DISK_UNDEFINED
#define DSM_RC_ABORT_BAD_DESTINATION         DSM_RS_ABORT_BAD_DESTINATION
#define DSM_RC_ABORT_DATAMOVER_NOT_AVAILABLE DSM_RS_ABORT_DATAMOVER_NOT_AVAILABLE
#define DSM_RC_ABORT_STGPOOL_COPY_CONT_NO    DSM_RS_ABORT_STGPOOL_COPY_CONT_NO
#define DSM_RC_ABORT_RETRY_SINGLE_TXN        DSM_RS_ABORT_RETRY_SINGLE_TXN
#define DSM_RC_ABORT_TOC_CREATION_FAIL       DSM_RS_ABORT_TOC_CREATION_FAIL
#define DSM_RC_ABORT_TOC_LOAD_FAIL           DSM_RS_ABORT_TOC_LOAD_FAIL
#define DSM_RC_ABORT_PATH_RESTRICTED         DSM_RS_ABORT_PATH_RESTRICTED
#define DSM_RC_ABORT_NO_LANFREE_SCRATCH      DSM_RS_ABORT_NO_LANFREE_SCRATCH
#define DSM_RC_ABORT_INSERT_NOT_ALLOWED      DSM_RS_ABORT_INSERT_NOT_ALLOWED
#define DSM_RC_ABORT_DELETE_NOT_ALLOWED      DSM_RS_ABORT_DELETE_NOT_ALLOWED
#define DSM_RC_ABORT_TXN_LIMIT_EXCEEDED      DSM_RS_ABORT_TXN_LIMIT_EXCEEDED
#define DSM_RC_ABORT_OBJECT_ALREADY_HELD     DSM_RS_ABORT_OBJECT_ALREADY_HELD
#define DSM_RC_ABORT_INVALID_CHUNK_REFERENCE DSM_RS_ABORT_INVALID_CHUNK_REFERENCE
#define DSM_RC_ABORT_DESTINATION_NOT_DEDUP   DSM_RS_ABORT_DESTINATION_NOT_DEDUP
#define DSM_RC_ABORT_DESTINATION_POOL_CHANGED DSM_RS_ABORT_DESTINATION_POOL_CHANGED
#define DSM_RC_ABORT_NOT_ROOT                DSM_RS_ABORT_NOT_ROOT


/* Definitions for server signon reject codes                        */
/* These error codes are in the range (51 to 99) inclusive.          */
#define DSM_RC_REJECT_NO_RESOURCES           51
#define DSM_RC_REJECT_VERIFIER_EXPIRED       52
#define DSM_RC_REJECT_ID_UNKNOWN             53
#define DSM_RC_REJECT_DUPLICATE_ID           54
#define DSM_RC_REJECT_SERVER_DISABLED        55
#define DSM_RC_REJECT_CLOSED_REGISTER        56
#define DSM_RC_REJECT_CLIENT_DOWNLEVEL       57
#define DSM_RC_REJECT_SERVER_DOWNLEVEL       58
#define DSM_RC_REJECT_ID_IN_USE              59
#define DSM_RC_REJECT_ID_LOCKED              61
#define DSM_RC_SIGNONREJECT_LICENSE_MAX      62
#define DSM_RC_REJECT_NO_MEMORY              63
#define DSM_RC_REJECT_NO_DB_SPACE            64
#define DSM_RC_REJECT_NO_LOG_SPACE           65
#define DSM_RC_REJECT_INTERNAL_ERROR         66
#define DSM_RC_SIGNONREJECT_INVALID_CLI      67 /* client type not licensed */
#define DSM_RC_CLIENT_NOT_ARCHRETPROT        68
#define DSM_RC_REJECT_LASTSESS_CANCELED      69
#define DSM_RC_REJECT_UNICODE_NOT_ALLOWED    70
#define DSM_RC_REJECT_NOT_AUTHORIZED         71
#define DSM_RC_REJECT_TOKEN_TIMEOUT          72
```

```
#define DSM_RC_REJECT_INVALID_NODE_TYPE      73
#define DSM_RC_REJECT_INVALID_SESSIONINIT    74
#define DSM_RC_REJECT_WRONG_PORT             75
#define DSM_RC_CLIENT_NOT_SPMRETPROT         79



#define DSM_RC_USER_ABORT          101 /* processing aborted by user        */
#define DSM_RC_NO_MEMORY           102 /* no RAM left to complete request   */
#define DSM_RC_TA_COMM_DOWN       2021 /* no longer used                    */
#define DSM_RC_FILE_NOT_FOUND      104 /* specified file not found          */
#define DSM_RC_PATH_NOT_FOUND      105 /* specified path doesn't exist      */
#define DSM_RC_ACCESS_DENIED       106 /* denied due to improper permission */
#define DSM_RC_NO_HANDLES          107 /* no more file handles available    */
#define DSM_RC_FILE_EXISTS         108 /* file already exists               */
#define DSM_RC_INVALID_PARM        109 /* invalid parameter passed. CRITICAL*/
#define DSM_RC_INVALID_HANDLE      110 /* invalid file handle passed        */
#define DSM_RC_DISK_FULL           111 /* out of disk space                 */
#define DSM_RC_PROTOCOL_VIOLATION  113 /* call protocol violation. CRITICAL */
#define DSM_RC_UNKNOWN_ERROR       114 /* unknown system error. CRITICAL    */
#define DSM_RC_UNEXPECTED_ERROR    115 /* unexpected error. CRITICAL        */
#define DSM_RC_FILE_BEING_EXECUTED 116 /* No write is allowed               */
#define DSM_RC_DIR_NO_SPACE        117 /* directory can't be expanded       */
#define DSM_RC_LOOPED_SYM_LINK     118 /* too many symbolic links were
                                          encountered in translating path.  */
#define DSM_RC_FILE_NAME_TOO_LONG  119 /* file name too long                */
#define DSM_RC_FILE_SPACE_LOCKED   120 /* filespace is locked by the system */
#define DSM_RC_FINISHED            121 /* finished processing               */
#define DSM_RC_UNKNOWN_FORMAT      122 /* unknown format                    */
#define DSM_RC_NO_AUTHORIZATION    123 /* server response when the client has
                                          no authorization to read another
                                          host's owner backup/archive data  */
#define DSM_RC_FILE_SPACE_NOT_FOUND 124/* specified file space not found    */
#define DSM_RC_TXN_ABORTED         125 /* transaction aborted               */
#define DSM_RC_SUBDIR_AS_FILE      126 /* Subdirectory name exists as file  */
#define DSM_RC_PROCESS_NO_SPACE    127 /* process has no more disk space.   */
#define DSM_RC_PATH_TOO_LONG       128 /* a directory path being build became
                                          too long                          */
#define DSM_RC_NOT_COMPRESSED      129 /* file thought to be compressed is
                                          actually not                      */
#define DSM_RC_TOO_MANY_BITS       130 /* file was compressed using more bits
                                          then the expander can handle      */
#define DSM_RC_SYSTEM_ERROR        131 /* internal system error             */
#define DSM_RC_NO_SERVER_RESOURCES 132 /* server out of resources.          */
#define DSM_RC_FS_NOT_KNOWN        133 /* the file space is not known by the
                                          server                            */
#define DSM_RC_NO_LEADING_DIRSEP   134 /* no leading directory separator    */
#define DSM_RC_WILDCARD_DIR        135 /* wildcard character in directory
                                          path when not allowed             */
#define DSM_RC_COMM_PROTOCOL_ERROR 136 /* communications protocol error     */
#define DSM_RC_AUTH_FAILURE        137 /* authentication failure            */
#define DSM_RC_TA_NOT_VALID        138 /* TA not a root and/or SUID program */
#define DSM_RC_KILLED              139 /* process killed.                   */

#define DSM_RC_RETRY               143 /* retry same operation again        */

#define DSM_RC_WOULD_BLOCK         145 /* operation would cause the system to
                                          block waiting for input.          */
#define DSM_RC_TOO_SMALL           146 /* area for compiled pattern small   */
#define DSM_RC_UNCLOSED            147 /* no closing bracket in pattern     */
#define DSM_RC_NO_STARTING_DELIMITER 148 /* pattern has to start with
                                            directory delimiter             */
#define DSM_RC_NEEDED_DIR_DELIMITER 149 /* a directory delimiter is needed
                                          immediately before and after the
                                          "match directories" metastring
                                          ("...") and one wasn't found      */
#define DSM_RC_UNKNOWN_FILE_DATA_TYPE 150 /* structured file data type is
                                             unknown                        */
#define DSM_RC_BUFFER_OVERFLOW     151 /* data buffer overflow              */


#define DSM_RC_NO_COMPRESS_MEMORY  154 /* Compress/Expand out of memory     */
#define DSM_RC_COMPRESS_GREW       155 /* Compression grew                  */
#define DSM_RC_INV_COMM_METHOD     156 /* Invalid comm method specified     */
#define DSM_RC_WILL_ABORT          157 /* Transaction will be aborted       */
#define DSM_RC_FS_WRITE_LOCKED     158 /* File space is write locked        */
#define DSM_RC_SKIPPED_BY_USER     159 /* User wanted file skipped in the
                                          case of ABORT_DATA_OFFLINE        */
#define DSM_RC_TA_NOT_FOUND        160 /* TA not found in it's directory    */
```

```
#define DSM_RC_TA_ACCESS_DENIED    161 /* Access to TA is denied           */
#define DSM_RC_FS_NOT_READY        162 /* File space not ready             */
#define DSM_RC_FS_IS_BAD           163 /* File space is bad                */
#define DSM_RC_FIO_ERROR           164 /* File input/output error          */
#define DSM_RC_WRITE_FAILURE       165 /* Error writing to file            */
#define DSM_RC_OVER_FILE_SIZE_LIMIT 166 /* File over system/user limit     */
#define DSM_RC_CANNOT_MAKE         167 /* Could not create file/directory,
                                          could be a bad name             */
#define DSM_RC_NO_PASS_FILE        168 /* password file needed and user is
                                          not root                        */
#define DSM_RC_VERFILE_OLD         169 /* password stored locally doesn't
                                          match the one at the host       */
#define DSM_RC_INPUT_ERROR         173 /* unable to read keyboard input    */
#define DSM_RC_REJECT_PLATFORM_MISMATCH 174 /* Platform name doesn't match
                                          up with what the server says
                                          is the platform for the client */
#define DSM_RC_TL_NOT_FILE_OWNER   175 /* User trying to backup a file is not
                                          the file's owner.               */
#define DSM_RC_COMPRESSED_DATA_CORRUPTED 176 /* Compressed data is corrupted    */
#define DSM_RC_UNMATCHED_QUOTE     177       /* missing starting or ending quote  */

#define DSM_RC_SIGNON_FAILOVER_MODE 178      /* Failed over to the replication server,
                                             running in failover mode */
#define DSM_RC_FAILOVER_MODE_FUNC_BLOCKED 179 /* function is blocked because
                                                session is in failover mode */

/*-----------------------------------------------------------------------*/
/* Return codes 180-199 are reserved for Policy Set handling              */
/*-----------------------------------------------------------------------*/
#define DSM_RC_PS_MULTBCG          181 /* Multiple backup copy groups in 1 MC*/
#define DSM_RC_PS_MULTACG          182 /* Multiple arch.  copy groups in 1 MC*/
#define DSM_RC_PS_NODFLTMC         183 /* Default MC name not in policy set  */
#define DSM_RC_TL_NOBCG            184 /* Backup req, no backup copy group   */
#define DSM_RC_TL_EXCLUDED         185 /* Backup req, excl. by in/ex filter  */
#define DSM_RC_TL_NOACG            186 /* Archive req, no archive copy group */
#define DSM_RC_PS_INVALID_ARCHMC   187 /* Invalid MC name in archive override*/
#define DSM_RC_NO_PS_DATA          188 /* No policy set data on the server   */
#define DSM_RC_PS_INVALID_DIRMC    189 /* Invalid directory MC specified in
                                          the options file.               */
#define DSM_RC_PS_NO_CG_IN_DIR_MC  190 /* No backup copy group in directory MC.
                                          Must specify an MC using DirMC
                                          option.                         */

#define DSM_RC_WIN32_UNSUPPORTED_FILE_TYPE 280 /* File is not of
                                             Win32 type FILE_TYPE_DISK  */

/*-----------------------------------------------------------------------*/
/* Return codes for the Trusted Communication Agent                       */
/*-----------------------------------------------------------------------*/
#define DSM_RC_TCA_NOT_ROOT        161 /* Access to TA is denied          */
#define DSM_RC_TCA_ATTACH_SHR_MEM_ERR  200 /* Error attaching shared memory  */
#define DSM_RC_TCA_SHR_MEM_BLOCK_ERR   200 /* Shared memory block error      */
#define DSM_RC_TCA_SHR_MEM_IN_USE      200 /* Shared memory block error      */
#define DSM_RC_TCA_SHARED_MEMORY_ERROR 200 /* Shared memory block error      */
#define DSM_RC_TCA_SEGMENT_MISMATCH    200 /* Shared memory block error      */
#define DSM_RC_TCA_FORK_FAILED     292 /* Error forking off TCA process      */
#define DSM_RC_TCA_DIED            294 /* TCA died unexpectedly              */
#define DSM_RC_TCA_INVALID_REQUEST 295 /* Invalid request sent to TCA        */
#define DSM_RC_TCA_SEMGET_ERROR    297 /* Error getting semaphores           */
#define DSM_RC_TCA_SEM_OP_ERROR    298 /* Error in semaphore set or wait     */
#define DSM_RC_TCA_NOT_ALLOWED     299 /* TCA not allowed (multi thread)     */

/*-----------------------------------------------------------------------*/
/* 400-430  for options                                                   */
/*-----------------------------------------------------------------------*/
#define DSM_RC_INVALID_OPT           400 /* invalid option                */
#define DSM_RC_NO_HOST_ADDR          405 /* Not enuf info to connect server  */
#define DSM_RC_NO_OPT_FILE           406 /* No default user configuration file*/
#define DSM_RC_MACHINE_SAME          408 /* -MACHINENAME same as real name   */
#define DSM_RC_INVALID_SERVER        409 /* Invalid server name from client  */
#define DSM_RC_INVALID_KEYWORD       410 /* Invalid option keyword           */
#define DSM_RC_PATTERN_TOO_COMPLEX   411 /* Can't match Include/Exclude entry*/
#define DSM_RC_NO_CLOSING_BRACKET    412 /* Missing closing bracket inc/excl */
#define DSM_RC_OPT_CLIENT_NOT_ACCEPTING 417/* Client doesn't accept this option
                                          from the server              */
#define DSM_RC_OPT_CLIENT_DOES_NOT_WANT 418/* Client doesn't want this value
                                          from the server              */
#define DSM_RC_OPT_NO_INCLEXCL_FILE  419   /* inclexcl file not found        */
#define DSM_RC_OPT_OPEN_FAILURE      420   /* can't open file                */
```

```
#define DSM_RC_OPT_INV_NODENAME      421/* used for Windows if nodename=local
                                         machine when CLUSTERNODE=YES       */
#define DSM_RC_OPT_NODENAME_INVALID  423/* generic invalid nodename           */
#define DSM_RC_OPT_ERRORLOG_CONFLICT 424/* both logmax & retention specified  */
#define DSM_RC_OPT_SCHEDLOG_CONFLICT 425/* both logmax & retention specified  */
#define DSM_RC_CANNOT_OPEN_TRACEFILE 426/* cannot open trace file             */
#define DSM_RC_CANNOT_OPEN_LOGFILE   427/* cannot open error log file         */
#define DSM_RC_OPT_SESSINIT_LF_CONFLICT 428/* both sessioninit=server and
               enablelanfree=yes are specified*/
#define DSM_RC_OPT_OPTION_IGNORE     429/* option will be ignored            */
#define DSM_RC_OPT_DEDUP_CONFLICT    430/* cannot open error log file        */
#define DSM_RC_OPT_HSMLOG_CONFLICT 431/* both logmax & retention specified  */


/*----------------------------------------------------------------------------*/
/* 600 to 610 for volume label codes                                          */
/*----------------------------------------------------------------------------*/
#define DSM_RC_DUP_LABEL           600 /* duplicate volume label found       */
#define DSM_RC_NO_LABEL            601 /* drive has no label                 */

/*----------------------------------------------------------------------------*/
/* Return codes for message file processing                                   */
/*----------------------------------------------------------------------------*/
#define DSM_RC_NLS_CANT_OPEN_TXT   610 /* error trying to open msg txt file  */
#define DSM_RC_NLS_CANT_READ_HDR   611 /* error trying to read header        */
#define DSM_RC_NLS_INVALID_CNTL_REC 612 /* invalid control record            */
#define DSM_RC_NLS_INVALID_DATE_FMT 613 /* invalid default date format       */
#define DSM_RC_NLS_INVALID_TIME_FMT 614 /* invalid default time format       */
#define DSM_RC_NLS_INVALID_NUM_FMT 615 /* invalid default number format      */

/*----------------------------------------------------------------------------*/
/* Return codes 620-630 are reserved for log message return codes             */
/*----------------------------------------------------------------------------*/
#define DSM_RC_LOG_CANT_BE_OPENED  620 /* error trying to open error log     */
#define DSM_RC_LOG_ERROR_WRITING_TO_LOG 621 /* error occurred writing to
                                       log file                            */
#define DSM_RC_LOG_NOT_SPECIFIED   622 /* no error log file was specified    */


/*----------------------------------------------------------------------------*/
/* Return codes 900-999 TSM CLIENT ONLY                                       */
/*----------------------------------------------------------------------------*/
#define DSM_RC_NOT_ADSM_AUTHORIZED  927 /* Must be ADSM authorized to perform*/
                                    /* action : root user or pwd auth    */
#define DSM_RC_REJECT_USERID_UNKNOWN 940 /* userid unknown on server          */
#define DSM_RC_FILE_IS_SYMLINK      959 /* errorlog or trace is a symbolic
                                       link
                                                                        */

#define DSM_RC_DIRECT_STORAGE_AGENT_UNSUPPORTED 961 /* Direct connection to SA not supported */
#define DSM_RC_FS_NAMESPACE_DOWNLEVEL 963 /* Long namespace has been removed from
                                     from the Netware volume */
#define DSM_RC_CONTINUE_NEW_CONSUMER   972 /* Continue processing using a new consumer */
#define DSM_RC_CONTINUE_NEW_CONSUMER_NODEDUP   973 /* Continue processing using a new consumer no dedup*/
#define DSM_RC_CONTINUE_NEW_CONSUMER_NOCOMPRESS 976 /* Continue processing using a new consumer no compression */


#define DSM_RC_SERVER_SUPPORTS_FUNC    994 /* the server supports this function  */
#define DSM_RC_SERVER_AND_SA_SUPPORT_FUNC 995 /* Both server and SA support func */
#define DSM_RC_SERVER_DOWNLEVEL_FUNC   996 /* The server is downlevel for func   */
#define DSM_RC_STORAGEAGENT_DOWNLEVEL  997 /* the storage agent is downlevel      */
#define DSM_RC_SERVER_AND_SA_DOWNLEVEL 998 /* both server and SA downlevel       */


/* TCP/IP error codes */
#define DSM_RC_TCPIP_FAILURE       -50 /* TCP/IP communications failure      */
#define DSM_RC_CONN_TIMEDOUT       -51 /* TCP/IP connection attempt timedout */
#define DSM_RC_CONN_REFUSED        -52 /* TCP/IP connection refused by host  */
#define DSM_RC_BAD_HOST_NAME       -53 /* TCP/IP invalid host name specified */
#define DSM_RC_NETWORK_UNREACHABLE -54 /* TCP/IP host name unreachable       */
#define DSM_RC_WINSOCK_MISSING     -55 /* TCP/IP WINSOCK.DLL missing         */
#define DSM_RC_TCPIP_DLL_LOADFAILURE -56 /* Error from LoadLibrary           */
#define DSM_RC_TCPIP_LOADFAILURE   -57 /* Error from GetProcAddress   */
#define DSM_RC_TCPIP_USER_ABORT    -58 /* User aborted while in TCP/IP layer */

/*----------------------------------------------------------------------------*/
/* Return codes (-71)-(-90) are reserved for CommTSM error codes              */
/*----------------------------------------------------------------------------*/
#define DSM_RC_TSM_FAILURE         -71 /* TSM communications failure         */
```

```
#define DSM_RC_TSM_ABORT           -72 /* Session aborted abnormally         */

/*comm3270 error codes - no longer used*/
#define DSM_RC_COMM_TIMEOUT       2021   /* no longer used                    */
#define DSM_RC_EMULATOR_INACTIVE  2021   /* no longer used                    */
#define DSM_RC_BAD_HOST_ID        2021   /* no longer used                    */
#define DSM_RC_HOST_SESS_BUSY     2021   /* no longer used                    */
#define DSM_RC_3270_CONNECT_FAILURE 2021 /* no longer used                    */
#define DSM_RC_NO_ACS3ELKE_DLL    2021   /* no longer used                    */
#define DSM_RC_EMULATOR_ERROR     2021   /* no longer used                    */
#define DSM_RC_EMULATOR_BACKLEVEL 2021   /* no longer used                    */
#define DSM_RC_CKSUM_FAILURE      2021   /* no longer used                    */


/* The following Return codes are for EHLLAPI for Windows                    */
#define DSM_RC_3270COMMError_DLL        2021   /* no longer used             */
#define DSM_RC_3270COMMError_GetProc    2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_DLL         2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_GetProc     2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_HostConnect 2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_AllocBuff   2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_SendKey     2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_PacketChk   2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_ChkSum      2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_HostTimeOut 2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_Send        2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_Recv        2021   /* no longer used             */
#define DSM_RC_EHLLAPIError_General     2021   /* no longer used             */
#define DSM_RC_PC3270_MISSING_DLL       2021   /* no longer used             */
#define DSM_RC_3270COMM_MISSING_DLL     2021   /* no longer used             */


/* NETBIOS error codes */
#define DSM_RC_NETB_ERROR         -151 /* Could not add node to LAN          */
#define DSM_RC_NETB_NO_DLL        -152 /* The ACSNETB.DLL could not be loaded*/
#define DSM_RC_NETB_LAN_ERR       -155 /* LAN error detected                 */
#define DSM_RC_NETB_NAME_ERR      -158 /* Netbios error on Add Name          */
#define DSM_RC_NETB_TIMEOUT       -159 /* Netbios send timeout               */
#define DSM_RC_NETB_NOTINST       -160 /* Netbios not installed - DOS        */
#define DSM_RC_NETB_REBOOT        -161 /* Netbios config err - reboot DOS    */

/* Named Pipe error codes */
#define DSM_RC_NP_ERROR                       -190

/* CPIC error codes */
#define DSM_RC_CPIC_ALLOCATE_FAILURE          2021 /* no longer used      */
#define DSM_RC_CPIC_TYPE_MISMATCH             2021 /* no longer used      */
#define DSM_RC_CPIC_PIP_NOT_SPECIFY_ERR       2021 /* no longer used      */
#define DSM_RC_CPIC_SECURITY_NOT_VALID        2021 /* no longer used      */
#define DSM_RC_CPIC_SYNC_LVL_NO_SUPPORT       2021 /* no longer used      */
#define DSM_RC_CPIC_TPN_NOT_RECOGNIZED        2021 /* no longer used      */
#define DSM_RC_CPIC_TP_ERROR                  2021 /* no longer used      */
#define DSM_RC_CPIC_PARAMETER_ERROR           2021 /* no longer used      */
#define DSM_RC_CPIC_PROD_SPECIFIC_ERR         2021 /* no longer used      */
#define DSM_RC_CPIC_PROGRAM_ERROR             2021 /* no longer used      */
#define DSM_RC_CPIC_RESOURCE_ERROR            2021 /* no longer used      */
#define DSM_RC_CPIC_DEALLOCATE_ERROR          2021 /* no longer used      */
#define DSM_RC_CPIC_SVC_ERROR                 2021 /* no longer used      */
#define DSM_RC_CPIC_PROGRAM_STATE_CHECK       2021 /* no longer used      */
#define DSM_RC_CPIC_PROGRAM_PARAM_CHECK       2021 /* no longer used      */
#define DSM_RC_CPIC_UNSUCCESSFUL              2021 /* no longer used      */
#define DSM_RC_UNKNOWN_CPIC_PROBLEM           2021 /* no longer used      */
#define DSM_RC_CPIC_MISSING_LU                2021 /* no longer used      */
#define DSM_RC_CPIC_MISSING_TP                2021 /* no longer used      */
#define DSM_RC_CPIC_SNA6000_LOAD_FAIL         2021 /* no longer used      */
#define DSM_RC_CPIC_STARTUP_FAILURE           2021 /* no longer used      */

/*---------------------------------------------------------------------------*/
/* Return codes -300 to -307 are reserved for IPX/SPX communications         */
/*---------------------------------------------------------------------------*/
#define DSM_RC_TLI_ERROR                      2021 /* no longer used      */
#define DSM_RC_IPXSPX_FAILURE                 2021 /* no longer used      */
#define DSM_RC_TLI_DLL_MISSING                2021 /* no longer used      */
#define DSM_RC_DLL_LOADFAILURE                2021 /* no longer used      */
#define DSM_RC_DLL_FUNCTION_LOADFAILURE       2021 /* no longer used      */
#define DSM_RC_IPXCONN_REFUSED                2021 /* no longer used      */
#define DSM_RC_IPXCONN_TIMEDOUT               2021 /* no longer used      */
#define DSM_RC_IPXADDR_UNREACHABLE            2021 /* no longer used      */
#define DSM_RC_CPIC_MISSING_DLL               2021 /* no longer used      */
```

```
#define DSM_RC_CPIC_DLL_LOADFAILURE            2021 /* no longer used      */
#define DSM_RC_CPIC_FUNC_LOADFAILURE           2021 /* no longer used      */

/*=== Shared Memory Protocol error codes   ===*/
#define DSM_RC_SHM_TCPIP_FAILURE               -450
#define DSM_RC_SHM_FAILURE                     -451
#define DSM_RC_SHM_NOTAUTH                     -452

#define DSM_RC_NULL_OBJNAME        2000 /* Object name pointer is NULL      */
#define DSM_RC_NULL_DATABLKPTR     2001 /* dataBlkPtr is NULL               */
#define DSM_RC_NULL_MSG            2002 /* msg parm in dsmRCMsg is NULL     */

#define DSM_RC_NULL_OBJATTRPTR     2004 /* Object Attr Pointer is NULL      */

#define DSM_RC_NO_SESS_BLK         2006 /* no server session info           */
#define DSM_RC_NO_POLICY_BLK       2007 /* no policy hdr     info           */
#define DSM_RC_ZERO_BUFLEN         2008 /* bufferLen is zero for dataBlkPtr */
#define DSM_RC_NULL_BUFPTR         2009 /* bufferPtr is NULL for dataBlkPtr */

#define DSM_RC_INVALID_OBJTYPE     2010 /* invalid object type              */
#define DSM_RC_INVALID_VOTE        2011 /* invalid vote                     */
#define DSM_RC_INVALID_ACTION      2012 /* invalid action                   */
#define DSM_RC_INVALID_DS_HANDLE   2014 /* invalid ADSM handle              */
#define DSM_RC_INVALID_REPOS       2015 /* invalid value for repository     */
#define DSM_RC_INVALID_FSNAME      2016 /* fs should start with dir delim   */
#define DSM_RC_INVALID_OBJNAME     2017 /* invalid full path name           */
#define DSM_RC_INVALID_LLNAME      2018 /* ll should start with dir delim   */
#define DSM_RC_INVALID_OBJOWNER    2019 /* invalid object owner name        */
#define DSM_RC_INVALID_ACTYPE      2020 /* invalid action type              */
#define DSM_RC_INVALID_RETCODE     2021 /* dsmRC in dsmRCMsg is invalid     */
#define DSM_RC_INVALID_SENDTYPE    2022 /* invalid send type                */
#define DSM_RC_INVALID_PARAMETER   2023 /* invalid parameter                */
#define DSM_RC_INVALID_OBJSTATE    2024 /* active, inactive, or any match?  */
#define DSM_RC_INVALID_MCNAME      2025 /* Mgmt class name not found        */
#define DSM_RC_INVALID_DRIVE_CHAR  2026 /* Drive letter is not alphabet     */
#define DSM_RC_NULL_FSNAME         2027 /* Filespace name is NULL           */
#define DSM_RC_INVALID_HLNAME      2028 /* hl should start with dir delim   */

#define DSM_RC_NUMOBJ_EXCEED       2029 /* BeginGetData num objs exceeded   */

#define DSM_RC_NEWPW_REQD          2030 /* new password is required         */
#define DSM_RC_OLDPW_REQD          2031 /* old password is required         */
#define DSM_RC_NO_OWNER_REQD       2032 /* owner not allowed. Allow default */
#define DSM_RC_NO_NODE_REQD        2033 /* node not allowed w/ pw=generate  */
#define DSM_RC_KEY_MISSING         2034 /* key file can't be found          */
#define DSM_RC_KEY_BAD             2035 /* content of key file is bad       */

#define DSM_RC_BAD_CALL_SEQUENCE   2041 /* Sequence of DSM calls not allowed*/
#define DSM_RC_INVALID_TSMBUFFER   2042 /* invalid value for tsmbuffhandle or dataPtr */
#define DSM_RC_TOO_MANY_BYTES      2043 /* too many bytes copied to buffer  */
#define DSM_RC_MUST_RELEASE_BUFFER 2044 /* cant exit app needs to release buffers */
#define DSM_RC_BUFF_ARRAY_ERROR    2045 /* internal buff array error        */
#define DSM_RC_INVALID_DATABLK     2046 /* using tsmbuff datablk should be null */
#define DSM_RC_ENCR_NOT_ALLOWED    2047 /* when using tsmbuffers encription not allowed */
#define DSM_RC_OBJ_COMPRESSED      2048 /* Can't restore using tsmBuff on compressed object */
#define DSM_RC_OBJ_ENCRYPTED       2049 /* Cant restore using tsmbuff an encr obj */
#define DSM_RC_WILDCHAR_NOTALLOWED 2050 /* Wild card not allowed for hl,ll  */
#define DSM_RC_POR_NOT_ALLOWED     2051 /* Can't use partial object restore with tsmBuffers */
#define DSM_RC_NO_ENCRYPTION_KEY   2052 /* Encryption key not found*/
#define DSM_RC_ENCR_CONFLICT       2053 /* mutually exclusive options */

#define DSM_RC_FSNAME_NOTFOUND     2060 /* Filespace name not found         */
#define DSM_RC_FS_NOT_REGISTERED   2061 /* Filespace name not registered    */
#define DSM_RC_FS_ALREADY_REGED    2062 /* Filespace already registered     */
#define DSM_RC_OBJID_NOTFOUND      2063 /* No object id to restore          */
#define DSM_RC_WRONG_VERSION       2064 /* Wrong level of code              */
#define DSM_RC_WRONG_VERSION_PARM  2065 /* Wrong level of parameter struct  */

#define DSM_RC_NEEDTO_ENDTXN       2070 /* Need to call dsmEndTxn           */

#define DSM_RC_OBJ_EXCLUDED        2080 /* Object is excluded by MC         */
#define DSM_RC_OBJ_NOBCG           2081 /* Object has no backup copy group  */
#define DSM_RC_OBJ_NOACG           2082 /* Object has no archive copy group */

#define DSM_RC_APISYSTEM_ERROR     2090 /* API internal error               */

#define DSM_RC_DESC_TOOLONG             2100 /* description is too long          */
#define DSM_RC_OBJINFO_TOOLONG          2101 /* object attr objinfo too long     */
#define DSM_RC_HL_TOOLONG               2102 /* High level qualifier is too long */
```

```
#define DSM_RC_PASSWD_TOOLONG           2103 /* password is too long            */
#define DSM_RC_FILESPACE_TOOLONG        2104 /* filespace name is too long      */
#define DSM_RC_LL_TOOLONG               2105 /* Low level qualifier is too long  */
#define DSM_RC_FSINFO_TOOLONG           2106 /* filespace length is too big     */
#define DSM_RC_SENDDATA_WITH_ZERO_SIZE  2107 /* send data w/ zero est           */

/*=== new return codes for dsmaccess ===*/
#define DSM_RC_INVALID_ACCESS_TYPE 2110 /* invalid access type             */
#define DSM_RC_QUERY_COMM_FAILURE  2111 /* communication error during query */
#define DSM_RC_NO_FILES_BACKUP     2112 /* No backed up files for this fs   */
#define DSM_RC_NO_FILES_ARCHIVE    2113 /* No archived files for this fs    */
#define DSM_RC_INVALID_SETACCESS   2114 /* invalid set access format        */

/*=== new return codes for dsmaccess ===*/
#define DSM_RC_STRING_TOO_LONG     2120 /* String parameter too long        */

#define DSM_RC_MORE_DATA           2200 /* There are more data to restore   */

#define DSM_RC_BUFF_TOO_SMALL      2210 /* DataBlk buffer too small for qry */

#define DSM_RC_NO_API_CONFIGFILE   2228 /*specified API confg file not found*/
#define DSM_RC_NO_INCLEXCL_FILE    2229 /* specified inclexcl file not found*/
#define DSM_RC_NO_SYS_OR_INCLEXCL  2230 /* either dsm.sys or inclexcl file
                                           specified in dsm.sys not found   */
#define DSM_RC_REJECT_NO_POR_SUPPORT 2231 /* server doesn't have POR support*/

#define DSM_RC_NEED_ROOT           2300 /* API caller must be root          */
#define DSM_RC_NEEDTO_CALL_BINDMC  2301 /* dsmBindMC must be called first   */
#define DSM_RC_CHECK_REASON_CODE   2302 /* check reason code from dsmEndTxn */
#define DSM_RC_NEEDTO_ENDTXN_DEDUP_SIZE_EXCEEDED 2303 /* max dedup bytes exceeded */

/*=== return codes 2400 - 2410 used by lic file see agentrc.h ===*/

/*=== return codes 2410 - 2430 used by Oracle agent see agentrc.h ===*/

#define DSM_RC_ENC_WRONG_KEY       4580 /* the key provided is incorrect    */
#define DSM_RC_ENC_NOT_AUTHORIZED  4582 /* user is not allowed to decrypt   */
#define DSM_RC_ENC_TYPE_UNKNOWN    4584 /* encryption type unknown          */

/*============================================================================
   Return codes (4600)-(4624) are reserved for clustering
============================================================================*/
#define DSM_RC_CLUSTER_INFO_LIBRARY_NOT_LOADED        4600
#define DSM_RC_CLUSTER_LIBRARY_INVALID                4601
#define DSM_RC_CLUSTER_LIBRARY_NOT_LOADED             4602
#define DSM_RC_CLUSTER_NOT_MEMBER_OF_CLUSTER          4603
#define DSM_RC_CLUSTER_NOT_ENABLED                    4604
#define DSM_RC_CLUSTER_NOT_SUPPORTED                  4605
#define DSM_RC_CLUSTER_UNKNOWN_ERROR                  4606


/*============================================================================
   Return codes (5701)-(5749) are reserved for proxy
============================================================================*/
#define DSM_RC_PROXY_REJECT_NO_RESOURCES              5702
#define DSM_RC_PROXY_REJECT_DUPLICATE_ID              5705
#define DSM_RC_PROXY_REJECT_ID_IN_USE                 5710
#define DSM_RC_PROXY_REJECT_INTERNAL_ERROR            5717
#define DSM_RC_PROXY_REJECT_NOT_AUTHORIZED            5722
#define DSM_RC_PROXY_INVALID_FROMNODE                 5746
#define DSM_RC_PROXY_INVALID_SERVERFREE               5747
#define DSM_RC_PROXY_INVALID_CLUSTER                  5748
#define DSM_RC_PROXY_INVALID_FUNCTION                 5749

/*============================================================================
   Return codes 5801 - 5849 are reserved for cryptography/security
============================================================================*/

#define DSM_RC_CRYPTO_ICC_ERROR                       5801
#define DSM_RC_CRYPTO_ICC_CANNOT_LOAD                 5802
#define DSM_RC_SSL_NOT_SUPPORTED                      5803
#define DSM_RC_SSL_INIT_FAILED                        5804
#define DSM_RC_SSL_KEYFILE_OPEN_FAILED                5805
#define DSM_RC_SSL_KEYFILE_BAD_PASSWORD               5806
#define DSM_RC_SSL_BAD_CERTIFICATE                    5807

/*============================================================================
   Return codes 6300 - 6399 are reserved for client-side deduplication
============================================================================*/
```

```
#define DSM_RC_DIGEST_VALIDATION_ERROR              6300  /* End-to-end digest validation err  */
#define DSM_RC_DATA_FINGERPRINT_ERROR               6301 /* Failure in Rabin fingeprinting    */
#define DSM_RC_DATA_DEDUP_ERROR                     6302 /* Error converting data into chunks */


#endif /* _H_DSMRC */
```

Related reference:

API return codes

Appendix B. API type definitions source files

This appendix contains structure definitions, type definitions, and constants for the API. The first header files, `dsmapitd.h` and `tsmapitd.h`, illustrate the definitions that are common to all operating systems.

The second header file, `dsmapips.h`, provides an example of definitions that are specific to a particular operating system; in this example, the Windows platform.

The third header file, `release.h`, includes the version and release information.

The information that is provided here contains a point-in-time copy of the files that are distributed with the API. View the files in the API distribution package for the latest version.

```
/*****************************************************************
* Tivoli Storage Manager                                         *
* API Client Component                                           *
*                                                                *
* (C) Copyright IBM Corporation 1993,2010                        *
*****************************************************************/

/*****************************************************************
* Header File Name: dsmapitd.h
*
* Environment:     ***********************************************
*                  ** This is a platform-independent source file **
*
*                  ***********************************************
*
* Design Notes:    This file contains basic data types and constants
*                  includable by all client source files. The constants
*                  within this file should be set properly for the
*                  particular machine and operating system on which the
*                  client software is to be run.
*
*                  Platform specific definitions are included in dsmapips.h
*
* Descriptive-name: Definitions for Tivoli Storage manager API constants
*----------------------------------------------------------------------*/

#ifndef _H_DSMAPITD
#define _H_DSMAPITD


#include "dsmapips.h"      /* Platform specific definitions*/
#include "release.h"

/*=== set the structure alignment to pack the structures ===*/
#if (_OPSYS_TYPE == DS_WINNT) && !defined(_WIN64)
#pragma pack(1)
#endif

#ifdef _MAC
/*=============================================================================
  choices are:
http://developer.apple.com/documentation/DeveloperTools/Conceptual/PowerPCRuntime/Data/chapter_2_section_3.html

#pragma option align=<mode>
where <mode> is power, mac68k, natural, or packed.
=============================================================================*/
#pragma options align=packed
#endif
```

© Copyright IBM Corp. 1993, 2016

```
typedef char osChar_t;

/*<><><>
```

```
/*-------------------------------------------------------------------------+
| Type definition for compressedState in QryResp                           |
+-------------------------------------------------------------------------*/
#define DSM_OBJ_COMPRESSED_UNKNOWN    0
#define DSM_OBJ_COMPRESSED_YES        1
#define DSM_OBJ_COMPRESSED_NO         2

/*-------------------------------------------------------------------------+
| Definitions for "group type" field in tsmGrouphandlerIn_t                |
+-------------------------------------------------------------------------*/

#define  DSM_GROUPTYPE_NONE         0x00   /* Not a group member           */
#define  DSM_GROUPTYPE_RESERVED1    0x01   /* for future use               */
#define  DSM_GROUPTYPE_PEER         0x02   /* Peer group                   */
#define  DSM_GROUPTYPE_RESERVED2    0x03   /* for future use               */

/*-------------------------------------------------------------------------+
| Definitions for "member type" field in tsmGrouphandlerIn_t               |
+-------------------------------------------------------------------------*/

#define  DSM_MEMBERTYPE_LEADER      0x01   /* group leader  */
#define  DSM_MEMBERTYPE_MEMBER      0x02   /* group member  */

/*-------------------------------------------------------------------------+
| Definitions for "operation type" field in tsmGrouphandlerIn_t            |
+-------------------------------------------------------------------------*/
#define DSM_GROUP_ACTION_BEGIN         0x01
#define DSM_GROUP_ACTION_OPEN          0x02 /* create new group             */
#define DSM_GROUP_ACTION_CLOSE         0x03 /* commit and save an open group */
#define DSM_GROUP_ACTION_ADD           0x04 /* Append to a group */
#define DSM_GROUP_ACTION_ASSIGNTO      0x05 /* Assign to a another group */
#define DSM_GROUP_ACTION_REMOVE        0x06 /* remove a member from a group */

/*-------------------------------------------------------------------------+
| Values for copySer in DetailCG structures for Query Mgmt Class response  |
+-------------------------------------------------------------------------*/
#define Copy_Serial_Static          1   /*Copy Serialization Static       */
#define Copy_Serial_Shared_Static   2   /*Copy Serialization Shared Static*/
#define Copy_Serial_Shared_Dynamic  3   /*Copy Serialization Shared Dynamic*/
#define Copy_Serial_Dynamic         4   /*Copy Serialization Dynamic      */

/*-------------------------------------------------------------------------+
| Values for copyMode in DetailCG structures for Query Mgmt Class response |
+-------------------------------------------------------------------------*/
#define Copy_Mode_Modified          1   /*Copy Mode Modified              */
#define Copy_Mode_Absolute          2   /*Copy Mode Absolute              */

/*-------------------------------------------------------------------------+
|  Values for objState in qryBackupData structure                          |
+-------------------------------------------------------------------------*/
#define DSM_ACTIVE            0x01      /* query only active objects      */
#define DSM_INACTIVE          0x02      /* query only inactive objects    */
#define DSM_ANY_MATCH         0xFF      /* query all backup objects       */

/*-------------------------------------------------------------------------+
| Boundary values for dsmDate.year field in qryArchiveData structure       |
+-------------------------------------------------------------------------*/
#define DATE_MINUS_INFINITE         0x0000      /* lowest boundary        */
#define DATE_PLUS_INFINITE          0xFFFF      /* highest upper boundary */

/*-------------------------------------------------------------------------+
| Bits masks for update action parameter on dsmUpdateFS()                  |
+-------------------------------------------------------------------------*/
#define DSM_FSUPD_FSTYPE              ((unsigned) 0x00000002)
#define DSM_FSUPD_FSINFO              ((unsigned) 0x00000004)
#define DSM_FSUPD_BACKSTARTDATE       ((unsigned) 0x00000008)
#define DSM_FSUPD_BACKCOMPLETEDATE    ((unsigned) 0x00000010)
#define DSM_FSUPD_OCCUPANCY           ((unsigned) 0x00000020)
#define DSM_FSUPD_CAPACITY            ((unsigned) 0x00000040)
```

```
#define DSM_FSUPD_RESERVED1           ((unsigned) 0x00000100)

/*-------------------------------------------------------------------------+
| Bits mask  for backup update action parameter on dsmUpdateObj()          |
+-------------------------------------------------------------------------*/
#define DSM_BACKUPD_OWNER              ((unsigned) 0x00000001)
#define DSM_BACKUPD_OBJINFO            ((unsigned) 0x00000002)
#define DSM_BACKUPD_MC                 ((unsigned) 0x00000004)

#define DSM_ARCHUPD_OWNER              ((unsigned) 0x00000001)
#define DSM_ARCHUPD_OBJINFO            ((unsigned) 0x00000002)
#define DSM_ARCHUPD_DESCR              ((unsigned) 0x00000004)

/*-------------------------------------------------------------------------+
|  Values for repository parameter on dsmDeleteFS()                        |
+-------------------------------------------------------------------------*/
#define DSM_ARCHIVE_REP       0x0A      /* archive repository            */
#define DSM_BACKUP_REP        0x0B      /* backup repository             */
#define DSM_REPOS_ALL         0x01      /* all respository types         */

/*-------------------------------------------------------------------------+
|  Values for vote parameter on dsmEndTxn()                                |
+-------------------------------------------------------------------------*/
#define DSM_VOTE_COMMIT  1             /* commit current transaction     */
#define DSM_VOTE_ABORT   2             /* roll back current transaction  */

/*-------------------------------------------------------------------------+
|  Values for various flags returned in ApiSessInfo structure.             |
+-------------------------------------------------------------------------*/
/* Client compression field codes */
#define COMPRESS_YES    1    /* client must compress data             */
#define COMPRESS_NO     2    /* client must NOT compress data         */
#define COMPRESS_CD     3    /* client determined                     */

/* Archive delete permission codes. */
#define ARCHDEL_YES     1    /* archive delete allowed                */
#define ARCHDEL_NO      2    /* archive delete NOT allowed            */

/* Backup delete permission codes. */
#define BACKDEL_YES     1    /* backup delete allowed                 */
#define BACKDEL_NO      2    /* backup delete NOT allowed             */


/*-------------------------------------------------------------------------+
  Values for various flags returned in optStruct structure.              |
-------------------------------------------------------------------------*/
#define DSM_PASSWD_GENERATE  1
#define DSM_PASSWD_PROMPT    0

#define DSM_COMM_TCP        1    /* tcpip         */
#define DSM_COMM_NAMEDPIPE  2    /* Named pipes   */
#define DSM_COMM_SHM        3    /* Shared Memory */

/* obsolete commmethods */
#define DSM_COMM_PVM_IUCV   12
#define DSM_COMM_3270       12
#define DSM_COMM_IUCV       12
#define DSM_COMM_PWSCS      12
#define DSM_COMM_SNA_LU6_2  12
#define DSM_COMM_IPXSPX     12    /* For IPX/SPX support */
#define DSM_COMM_NETBIOS    12    /* NETBIOS */
#define DSM_COMM_400COMM    12
#define DSM_COMM_CLIO       12    /* CLIO/S */
/*-------------------------------------------------------------------------+
|  Values for userNameAuthorities in dsmInitEx for future use              |
+-------------------------------------------------------------------------*/
#define DSM_USERAUTH_NONE     ((dsInt16_t)0x0000)
#define DSM_USERAUTH_ACCESS   ((dsInt16_t)0x0001)
#define DSM_USERAUTH_OWNER    ((dsInt16_t)0x0002)
#define DSM_USERAUTH_POLICY   ((dsInt16_t)0x0004)
```

```
#define DSM_USERAUTH_SYSTEM   ((dsInt16_t)0x0008)

/*-------------------------------------------------------------------------+
|  Values for encryptionType on dsmEndSendObjEx, queryResp                 |
+-------------------------------------------------------------------------*/
#define DSM_ENCRYPT_NO              ((dsUint8_t)0x00)
#define DSM_ENCRYPT_USER            ((dsUint8_t)0x01)
#define DSM_ENCRYPT_CLIENTENCRKEY   ((dsUint8_t)0x02)
#define DSM_ENCRYPT_DES_56BIT       ((dsUint8_t)0x04)
#define DSM_ENCRYPT_AES_128BIT      ((dsUint8_t)0x08)
#define DSM_ENCRYPT_AES_256BIT      ((dsUint8_t)0x10)

/*-------------------------------------------------------------------------+
| Definitions for mediaClass field.                                        |
+-------------------------------------------------------------------------*/
/*
 *  The following constants define a hierarchy of media access classes.
 *  Lower numbers indicate media which can supply faster access to data.
 */

/* Fixed: represents the class of on-line, fixed media (such as
          hard disks). */
#define  MEDIA_FIXED          0x10

/* Library: represents the class of mountable media accessible
            through a mechanical mounting device. */
#define  MEDIA_LIBRARY        0x20

/* future use */
#define  MEDIA_NETWORK        0x30

/* future use */
#define  MEDIA_SHELF          0x40

/* future use */
#define  MEDIA_OFFSITE        0x50

/* future use */
#define  MEDIA_UNAVAILABLE    0xF0


/*-------------------------------------------------------------------------+
| Type definition for partial object data for dsmBeginGetData()            |
+-------------------------------------------------------------------------*/
typedef struct
{
    dsUint16_t     stVersion;          /* Structure version                    */
    dsStruct64_t   partialObjOffset;   /* offset into object to begin reading  */
    dsStruct64_t   partialObjLength;   /* amount of object to read             */
} PartialObjData ;               /* partial object data                  */

#define PartialObjDataVersion 1  /*                                      */

/*-------------------------------------------------------------------------+
|  Type definition for date structure                                      |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t    year;                 /* year, 16-bit integer (e.g., 1990) */
   dsUint8_t     month;                /* month, 8-bit integer (1 - 12)     */
   dsUint8_t     day;                  /* day. 8-bit integer (1 - 31)       */
   dsUint8_t     hour;                 /* hour, 8-bit integer (0 - 23)      */
   dsUint8_t     minute;               /* minute, 8-bit integer (0 - 59)    */
   dsUint8_t     second;               /* second, b-bit integer (0 - 59)    */
}dsmDate ;

/*-------------------------------------------------------------------------+
|  Type definition for Object ID on dsmGetObj() and in dsmGetList structure|
+-------------------------------------------------------------------------*/
typedef dsStruct64_t  ObjID ;
```

```
/*-------------------------------------------------------------------------+
| Type definition for dsmQueryBuff on dsmBeginQuery()                      |
+-------------------------------------------------------------------------*/
typedef void dsmQueryBuff ;

/*-------------------------------------------------------------------------+
| Type definition for dsmGetType parameter on dsmBeginGetData()            |
+-------------------------------------------------------------------------*/
typedef enum
{
        gtBackup = 0x00,                    /* Backup processing type    */
        gtArchive                           /* Archive processing type   */
} dsmGetType ;

/*-------------------------------------------------------------------------+
|  Type definition for dsmQueryType parameter on dsmBeginQuery()           |
+-------------------------------------------------------------------------*/
typedef enum
{
   qtArchive = 0x00,                /* Archive query type                */
   qtBackup,                        /* Backup query type                 */
   qtBackupActive,                  /* Fast query for active backup files */
   qtFilespace,                     /* Filespace query type              */
   qtMC,                            /* Mgmt. class query type            */
   qtReserved1,                     /* future use                        */
   qtReserved2,                     /* future use                        */
   qtReserved3,                     /* future use                        */
   qtReserved4,                     /* future use                        */
   qtBackupGroups,                  /* group leaders in a specific fs    */
   qtOpenGroups,                    /* Open groups in a specific fs      */
   qtReserved5,                     /* future use                        */
   qtProxyNodeAuth,                 /* nodes that his node can proxy to  */
   qtProxyNodePeer,                 /* Peer nodes with the same target   */
   qtReserved6,                     /* future use                        */
   qtReserved7,                     /* future use                        */
   qtReserved8                      /* future use                        */
}dsmQueryType ;

/*-------------------------------------------------------------------------+
|  Type definition sendType parameter on dsmBindMC() and dsmSendObj()      |
+-------------------------------------------------------------------------*/
typedef enum
{
   stBackup = 0x00,                        /* Backup processing type    */
   stArchive,                              /* Archive processing type   */
   stBackupMountWait,           /* Backup processing with mountwait on   */
   stArchiveMountWait           /* Archive processing with mountwait on  */
}dsmSendType ;

/*-------------------------------------------------------------------------+
|  Type definition for delType parameter on dsmDeleteObj()                 |
+-------------------------------------------------------------------------*/
typedef enum
{
   dtArchive = 0x00,                             /* Archive delete type */
   dtBackup,                          /* Backup delete (deactivate) type */
   dtBackupID                        /* Backup delete (remove)     type */
}dsmDelType ;

/*-------------------------------------------------------------------------+
|  Type definition sendType parameter on dsmSetAccess()                    |
+-------------------------------------------------------------------------*/
typedef enum
{
   atBackup = 0x00,                        /* Backup processing type    */
   atArchive                              /* Archive processing type   */
}dsmAccessType;

/*-------------------------------------------------------------------------+
```

```
|  Type definition for API Version on dsmInit() and dsmQueryApiVersion()   |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t version;       /* API version                     */
   dsUint16_t release;       /* API release                     */
   dsUint16_t level;         /* API level                       */
}dsmApiVersion;

/*----------------------------------------------------------------------------+
|  Type definition for API Version on dsmInit() and dsmQueryApiVersion()   |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t stVersion;     /* Structure version               */
   dsUint16_t version;       /* API version                     */
   dsUint16_t release;       /* API release                     */
   dsUint16_t level;         /* API level                       */
   dsUint16_t subLevel;      /* API sub level                   */
   dsmBool_t  unicode;       /* API unicode?                    */
}dsmApiVersionEx;


#define apiVersionExVer       2

/*----------------------------------------------------------------------------+
|  Type definition for Application Version on dsmInit()                      |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t   stVersion;            /* Structure version            */
   dsUint16_t   applicationVersion;   /* application version number   */
   dsUint16_t   applicationRelease;   /* application release number   */
   dsUint16_t   applicationLevel;     /* application level number     */
   dsUint16_t   applicationSubLevel;  /* application sub level number */
} dsmAppVersion;

#define appVersionVer      1

/*----------------------------------------------------------------------------+
|  Type definition for object name used on BindMC, Send, Delete, Query     |
+--------------------------------------------------------------------------*/
typedef struct S_dsmObjName
{
   char       fs[DSM_MAX_FSNAME_LENGTH + 1] ;              /* Filespace name */
   char       hl[DSM_MAX_HL_LENGTH + 1] ;                 /* High level name */
   char       ll[DSM_MAX_LL_LENGTH + 1] ;                  /* Low level name */
   dsUint8_t  objType;          /* for object type values, see defines above */
}dsmObjName;


/*----------------------------------------------------------------------------+
|  Type definition for Backup delete info on dsmDeleteObj()                  |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                   /* structure version      */
   dsmObjName       *objNameP ;                   /* object name            */
   dsUint32_t       copyGroup ;                   /* copy group             */
}delBack ;

#define  delBackVersion    1

/*----------------------------------------------------------------------------+
|  Type definition for Archive delete info on dsmDeleteObj()                 |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                   /* structure version      */
   dsStruct64_t       objId ;                       /* object ID              */
```

```
}delArch ;

#define  delArchVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for Backup ID delete info on dsmDeleteObj()             |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                   /* structure version      */
   dsStruct64_t       objId ;                       /* object ID              */
}delBackID;

#define  delBackIDVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for delete info on dsmDeleteObj()                       |
+-------------------------------------------------------------------------*/
typedef union
{
   delBack   backInfo ;
   delArch   archInfo ;
   delBackID backIDInfo ;
}dsmDelInfo ;

/*-------------------------------------------------------------------------+
|  Type definition for Object Attribute parameter on dsmSendObj()          |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t   stVersion;                      /* Structure version */
   char         owner[DSM_MAX_OWNER_LENGTH + 1]; /* object owner */
   dsStruct64_t sizeEstimate;                   /* Size estimate in bytes of the object */
   dsmBool_t    objCompressed;                  /* Is object already compressed? */
   dsUint16_t   objInfoLength;                  /* length of object-dependent info */
   char         *objInfo;                       /* object-dependent info */
   char         *mcNameP;                       /* mgmnt class name for override  */
   dsmBool_t    disableDeduplication;           /* force no dedup for this object */
   dsmBool_t    useExtObjInfo;                  /* use ext obj info up to 1536 */
}ObjAttr;

#define ObjAttrVersion  4



/*-------------------------------------------------------------------------+
| Type definition for mcBindKey returned on dsmBindMC()                    |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t  stVersion;                   /* structure version          */
   char        mcName[DSM_MAX_MC_NAME_LENGTH + 1];
                                        /* Name of mc bound to object. */
   dsmBool_t   backup_cg_exists;                            /* True/false */
   dsmBool_t   archive_cg_exists;                           /* True/false */
   char        backup_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
                                             /* Backup copy dest. name */
   char        archive_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
                                               /* Arch copy dest.name */
}mcBindKey;

#define mcBindKeyVersion  1


/*-------------------------------------------------------------------------+
| Type definition for object list on dsmBeginGetData()                     |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;         /* structure version            */
```

```
   dsUint32_t       numObjId ;          /* number of object IDs in the list */
   ObjID           *objId ;             /* list of object IDs to restore*/
   PartialObjData  *partialObjData;     /*list of partial obj data info */
}dsmGetList ;

#define dsmGetListVersion     2  /* default if not using Partial Obj data */
#define dsmGetListPORVersion  3  /* version if using Partial Obj data     */

/*-------------------------------------------------------------------------+
|  Type definition for DataBlk used to Get or Send data                    |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t stVersion ;               /* structure version            */
   dsUint32_t bufferLen;                /* Length of buffer passed below */
   dsUint32_t numBytes;                 /* Actual number of bytes read from */
                                        /* or written to the buffer */
   char       *bufferPtr;               /* Data buffer */
   dsUint32_t numBytesCompressed;       /* on send actual bytes compressed */
   dsUint16_t reserved;                 /* for future use                  */
}DataBlk;

#define DataBlkVersion  3


/*-------------------------------------------------------------------------+
|  Type definition for Mgmt Class queryBuffer on dsmBeginQuery()           |
+-------------------------------------------------------------------------*/
typedef struct S_qryMCData
{
   dsUint16_t   stVersion;                            /* structure version */
   char         *mcName;                        /* Mgmt class name */
                        /* single name to get one or empty string to get all*/
   dsmBool_t    mcDetail;                           /* Want details or not? */
}qryMCData;

#define qryMCDataVersion  1

/*=== values for RETINIT ===*/
#define ARCH_RETINIT_CREATE 0
#define ARCH_RETINIT_EVENT  1

/*-------------------------------------------------------------------------+
|  Type definition for Archive Copy Group details on Query MC response     |
+-------------------------------------------------------------------------*/
typedef struct S_archDetailCG
{
   char         cgName[DSM_MAX_CG_NAME_LENGTH + 1];       /* Copy group name */
   dsUint16_t   frequency;                       /* Copy (archive) frequency */
   dsUint16_t   retainVers;                            /* Retain version */
   dsUint8_t    copySer;       /* for copy serialization values, see defines */
   dsUint8_t    copyMode;         /* for copy mode values, see defines above */
   char         destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
   dsmBool_t    bLanFreeDest;           /* Destination has lan free path?   */
   dsmBool_t    reserved;               /* Not currently used               */
   dsUint8_t    retainInit;             /* possible values see above        */
   dsUint16_t   retainMin;              /* if retInit is EVENT num of days  */
   dsmBool_t    bDeduplicate;           /* destination has dedup enabled    */
}archDetailCG;

/*-------------------------------------------------------------------------+
|  Type definition for Backup Copy Group details on Query MC response      |
+-------------------------------------------------------------------------*/
typedef struct S_backupDetailCG
{
   char         cgName[DSM_MAX_CG_NAME_LENGTH + 1];       /* Copy group name */
   dsUint16_t   frequency;                             /* Backup frequency */
   dsUint16_t   verDataExst;                         /* Versions data exists */
   dsUint16_t   verDataDltd;                        /* Versions data deleted */
   dsUint16_t   retXtraVers;                        /* Retain extra versions */
```

```
   dsUint16_t   retOnlyVers;                        /* Retain only versions */
   dsUint8_t    copySer;       /* for copy serialization values, see defines */
   dsUint8_t    copyMode;         /* for copy mode values, see defines above */
   char         destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
   dsmBool_t    bLanFreeDest;            /* Destination has lan free path?   */
   dsmBool_t    reserved;                /* Not currently used               */
   dsmBool_t    bDeduplicate;            /* destination has dedup enabled    */
}backupDetailCG;


/*-------------------------------------------------------------------------+
|  Type definition for Query Mgmt Class detail response on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct S_qryRespMCDetailData
{
   dsUint16_t      stVersion;                      /* structure version */
   char            mcName[DSM_MAX_MC_NAME_LENGTH + 1];        /* mc name */
   char            mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1]; /*mc description */
   archDetailCG    archDet;                 /* Archive copy group detail */
   backupDetailCG  backupDet;                /* Backup copy group detail */
}qryRespMCDetailData;

#define qryRespMCDetailDataVersion  4

/*-------------------------------------------------------------------------+
| Type definition for Query Mgmt Class summary response on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/
typedef struct S_qryRespMCData
{
   dsUint16_t   stVersion;                              /* structure version */
   char         mcName[DSM_MAX_MC_NAME_LENGTH + 1];              /* mc name */
   char         mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1];       /* mc description */
}qryRespMCData;

#define qryRespMCDataVersion  1



/*-------------------------------------------------------------------------+
|  Type definition for Archive queryBuffer on dsmBeginQuery()              |
+-------------------------------------------------------------------------*/
typedef struct S_qryArchiveData
{
   dsUint16_t   stVersion;                         /* structure version */
   dsmObjName   *objName;                     /* Full dsm name of object */
   char         *owner;                                /* owner name */
                /* for maximum date boundaries, see defines above */
   dsmDate      insDateLowerBound;      /* low bound archive insert date */
   dsmDate      insDateUpperBound;       /* hi bound archive insert date */
   dsmDate      expDateLowerBound;          /* low bound expiration date */
   dsmDate      expDateUpperBound;           /* hi bound expiration date */
   char         *descr;               /* archive description */
} qryArchiveData;

#define qryArchiveDataVersion  1



/*=== values for retentionInitiated field ===*/
#define DSM_ARCH_RETINIT_UNKNOWN 0 /* ret init is unknown (down-level srv)  */
#define DSM_ARCH_RETINIT_STARTED 1 /* retention clock is started            */
#define DSM_ARCH_RETINIT_PENDING 2 /* retention clock is not started        */


/*=== Values for objHeld ===*/
#define DSM_ARCH_HELD_UNKNOWN 0    /* unknown hold status (down-level srv)  */
#define DSM_ARCH_HELD_FALSE   1    /* object is NOT in a delete hold state  */
#define DSM_ARCH_HELD_TRUE    2    /* object is in a delete hold state      */

/*-------------------------------------------------------------------------+
```

```
| Type definition for Query Archive response on dsmGetNextQObj()          |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespArchiveData
{
   dsUint16_t      stVersion;                          /* structure version */
   dsmObjName      objName;                      /* Filespace name qualifier */
   dsUint32_t      copyGroup;                          /* copy group number */
   char            mcName[DSM_MAX_MC_NAME_LENGTH + 1];           /* mc name */
   char            owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
   dsStruct64_t    objId;                                 /* Unique copy id */
   dsStruct64_t    reserved;                        /* backward compatability */
   dsUint8_t       mediaClass;                          /* media access class */
   dsmDate         insDate;                        /* archive insertion date */
   dsmDate         expDate;                    /* expiration date for object */
   char            descr[DSM_MAX_DESCR_LENGTH + 1];   /* archive description */
   dsUint16_t      objInfolen;             /* length of object-dependent info*/
   char            reservedObjInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
   dsUint160_t     restoreOrderExt;                       /* restore order */
   dsStruct64_t    sizeEstimate;             /* size estimate stored by user*/
   dsUint8_t       compressType;                         /* Compression flag*/
   dsUint8_t       retentionInitiated;   /* object waiting on retention event*/
   dsUint8_t       objHeld;  /*object is on retention "hold" see values above*/
   dsUint8_t       encryptionType;                    /* type of encryption */
   dsmBool_t       clientDeduplicated;             /* obj deduplicated by API*/
   char            objInfo[DSM_MAX_EXT_OBJINFO_LENGTH];  /*object-dependent info */
   char            compressAlg[DSM_MAX_COMPRESSTYPE_LENGTH + 1];  /* compression algorithm name */
}qryRespArchiveData;

#define qryRespArchiveDataVersion  7

/*-------------------------------------------------------------------------+
|  Type definition for Archive sendBuff parameter on dsmSendObj()          |
+-------------------------------------------------------------------------*/
typedef struct S_sndArchiveData
{
   dsUint16_t   stVersion;                  /* structure version */
   char         *descr;                   /* archive description */
}sndArchiveData;

#define sndArchiveDataVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for Backup queryBuffer on dsmBeginQuery()               |
+-------------------------------------------------------------------------*/
typedef struct S_qryBackupData
{
   dsUint16_t  stVersion;         /* structure version */
   dsmObjName  *objName;          /* full dsm name of object */
   char        *owner;            /* owner name */
   dsUint8_t   objState;          /* object state selector */
   dsmDate     pitDate;           /* Date value for point in time restore */
                                  /* for possible values, see defines above */
}qryBackupData;

#define qryBackupDataVersion  2

typedef struct
{
  dsUint8_t    reserved1;
  dsStruct64_t reserved2;
} reservedInfo_t;                 /* for future use */

/*-------------------------------------------------------------------------+
| Type definition for Query Backup response on dsmGetNextQObj()            |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespBackupData
{
   dsUint16_t      stVersion;                          /* structure version */
   dsmObjName      objName;                     /* full dsm name of object */
   dsUint32_t      copyGroup;                          /* copy group number */
```

```
   char            mcName[DSM_MAX_MC_NAME_LENGTH + 1];           /* mc name */
   char            owner[DSM_MAX_OWNER_LENGTH + 1];           /* owner name */
   dsStruct64_t    objId;                                 /* Unique object id */
   dsStruct64_t    reserved;                        /* backward compatability */
   dsUint8_t       mediaClass;                          /* media access class */
   dsUint8_t       objState;                       /* Obj state, active, etc. */
   dsmDate         insDate;                         /* backup insertion date */
   dsmDate         expDate;                     /* expiration date for object */
   dsUint16_t      objInfolen;             /* length of object-dependent info*/
   char            reservedObjInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
   dsUint160_t     restoreOrderExt;                        /* restore order */
   dsStruct64_t    sizeEstimate;             /* size estimate stored by user */
   dsStruct64_t    baseObjId;
   dsUint16_t      baseObjInfolen;             /* length of base object-dependent info*/
   dsUint8_t       baseObjInfo[DSM_MAX_OBJINFO_LENGTH];  /* base object-dependent info */
   dsUint160_t     baseRestoreOrder;                        /* restore order */
   dsUint32_t      fsID;
   dsUint8_t       compressType;
   dsmBool_t       isGroupLeader;
   dsmBool_t       isOpenGroup;
   dsUint8_t       reserved1;               /* for future use */
   dsmBool_t       reserved2;               /* for future use */
   dsUint16_t      reserved3;               /* for future use */
   reservedInfo_t  *reserved4;              /* for future use */
   dsUint8_t       encryptionType;          /* type of encryption */
   dsmBool_t       clientDeduplicated;      /* obj deduplicated by API*/
   char            objInfo[DSM_MAX_EXT_OBJINFO_LENGTH];  /*object-dependent info */
   char            compressAlg[DSM_MAX_COMPRESSTYPE_LENGTH + 1];  /* compression algorithm name */
}qryRespBackupData;

#define qryRespBackupDataVersion  8

/*--------------------------------------------------------------------------+
|  Type definition for Active Backup queryBuffer on dsmBeginQuery()
|
|  Notes:  For the active backup query, only the fs (filespace) and objType
|          fields of objName need be set.  objType can only be set to
|          DSM_OBJ_FILE or DSM_OBJ_DIRECTORY.  DSM_OBJ_ANY_TYPE will not
|          find a match on the query.
+--------------------------------------------------------------------------*/
typedef struct S_qryABackupData
{
   dsUint16_t      stVersion;                          /* structure version */
   dsmObjName      *objName;                    /* Only fs and objtype used */
}qryABackupData;

#define qryABackupDataVersion  1

/*--------------------------------------------------------------------------+
| Type definition for Query Active Backup response on dsmGetNextQObj()     |
+--------------------------------------------------------------------------*/
typedef struct S_qryARespBackupData
{
   dsUint16_t  stVersion;                          /* structure version */
   dsmObjName  objName;                       /* full dsm name of object */
   dsUint32_t  copyGroup;                          /* copy group number */
   char        mcName[DSM_MAX_MC_NAME_LENGTH + 1];/*management class name*/
   char        owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
   dsmDate     insDate;                         /* backup insertion date */
   dsUint16_t  objInfolen;             /* length of object-dependent info*/
   char        reservedObjInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
   char        objInfo[DSM_MAX_EXT_OBJINFO_LENGTH];  /*object-dependent info */
}qryARespBackupData;

#define qryARespBackupDataVersion  2

/*--------------------------------------------------------------------------+
|  Type definition for Backup queryBuffer on dsmBeginQuery()                |
+--------------------------------------------------------------------------*/
typedef struct qryBackupGroups
```

```
{
   dsUint16_t   stVersion;         /* structure version */
   dsUint8_t    groupType;
   char         *fsName;
   char         *owner;
   dsStruct64_t groupLeaderObjId;
   dsUint8_t    objType;
   dsmBool_t    noRestoreOrder;
   dsmBool_t    noGroupInfo;
   char         *hl;
}qryBackupGroups;

#define qryBackupGroupsVersion 3


/*-------------------------------------------------------------------------+
|  Type definition for proxynode queryBuffer on dsmBeginQuery()            |
+-------------------------------------------------------------------------*/
typedef struct qryProxyNodeData
{
   dsUint16_t  stVersion;                  /* structure version */
   char        *targetNodeName;            /* target node name  */
}qryProxyNodeData;

#define qryProxyNodeDataVersion  1

/*-------------------------------------------------------------------------+
| Type definition for qryRespProxyNodeData  parameter used on dsmGetNextQObj()|
+-------------------------------------------------------------------------*/

typedef struct
{
   dsUint16_t      stVersion ;                         /* structure version */
   char            targetNodeName[DSM_MAX_ID_LENGTH+1]; /* target node name  */
   char            peerNodeName[DSM_MAX_ID_LENGTH+1];   /* Peer node name    */
   char            hlAddress[DSM_MAX_ID_LENGTH+1];      /* peer hlAddress    */
   char            llAddress[DSM_MAX_ID_LENGTH+1];      /* peer hlAddress    */
}qryRespProxyNodeData;

#define qryRespProxyNodeDataVersion  1


/*-------------------------------------------------------------------------+
|  Type definition for WINNT and OS/2 Filespace attributes                 |
+-------------------------------------------------------------------------*/
typedef struct
{
   char         driveLetter ;          /* drive letter for filespace    */
   dsUint16_t   fsInfoLength;          /* fsInfo length used            */
   char         fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
}dsmDosFSAttrib ;

/*-------------------------------------------------------------------------+
|  Type definition for UNIX Filespace attributes                           |
+-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t   fsInfoLength;          /* fsInfo length used            */
   char         fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
}dsmUnixFSAttrib ;

/*-------------------------------------------------------------------------+
|  Type definition for NetWare Filespace attributes                        |
+-------------------------------------------------------------------------*/
typedef dsmUnixFSAttrib dsmNetwareFSAttrib;

/*-------------------------------------------------------------------------+
|  Type definition for Filespace attributes on all Filespace calls         |
+-------------------------------------------------------------------------*/
typedef union
```

```
{
   dsmNetwareFSAttrib  netwareFSAttr;
   dsmUnixFSAttrib     unixFSAttr ;
   dsmDosFSAttrib      dosFSAttr ;
}dsmFSAttr ;

/*-------------------------------------------------------------------------+
|  Type definition for fsUpd parameter on dsmUpdateFS()
+-------------------------------------------------------------------------*/
typedef struct S_dsmFSUpd
{
   dsUint16_t      stVersion ;             /* structure version             */
   char            *fsType ;               /* filespace type                */
   dsStruct64_t    occupancy ;             /* occupancy estimate            */
   dsStruct64_t    capacity  ;             /* capacity estimate             */
   dsmFSAttr       fsAttr ;                /* platform specific attributes  */
}dsmFSUpd ;

#define dsmFSUpdVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for Filespace queryBuffer on dsmBeginQuery()            |
+-------------------------------------------------------------------------*/
typedef struct S_qryFSData
{
   dsUint16_t  stVersion;                /* structure version */
   char        *fsName;                  /* File space name */
}qryFSData;

#define qryFSDataVersion  1

/*-------------------------------------------------------------------------+
| Type definition for Query Filespace response on dsmGetNextQObj()         |
+-------------------------------------------------------------------------*/
typedef struct S_qryRespFSData
{
   dsUint16_t     stVersion;                              /* structure version */
   char           fsName[DSM_MAX_FSNAME_LENGTH + 1];         /* Filespace name */
   char           fsType[DSM_MAX_FSTYPE_LENGTH + 1] ;        /* Filespace type */
   dsStruct64_t   occupancy;                           /* Occupancy est. in bytes.*/
   dsStruct64_t   capacity;                            /* Capacity est. in bytes. */
   dsmFSAttr      fsAttr ;                    /* platform specific attributes  */
   dsmDate        backStartDate;              /* start backup date             */
   dsmDate        backCompleteDate;           /* end backup Date               */
   dsmDate        reserved1;                  /* For future use                */
   dsmDate        lastReplStartDate;        /* The last time replication was started */
   dsmDate        lastReplCmpltDate;        /* The last time replication completed   */
                                            /*   (could have had a failure,          */
                                            /*    but it still completes)            */
   dsmDate        lastBackOpDateFromServer; /* The last store time stamp the client  */
                                            /*    saved on the server                */
   dsmDate        lastArchOpDateFromServer; /* The last store time stamp the client  */
                                            /*    saved on the server                */
   dsmDate        lastSpMgOpDateFromServer; /* The last store time stamp the client  */
                                            /*    saved on the server                */
   dsmDate        lastBackOpDateFromLocal;  /* The last store time stamp the client  */
                                            /*    saved on the Local                 */
   dsmDate        lastArchOpDateFromLocal;  /* The last store time stamp the client  */
                                            /*    saved on the Local                 */
   dsmDate        lastSpMgOpDateFromLocal;  /* The last store time stamp the client  */
                                            /*    saved on the Local                 */
   dsInt32_t      failOverWriteDelay;       /* Minutes for client to wait before  allowed  */
                                            /* to store to this Repl srvr, Specail codes:  */
                                            /* NO_ACCESS(-1), ACCESS_RDONLY (-2)           */
}qryRespFSData;

#define qryRespFSDataVersion 4

/*-------------------------------------------------------------------------+
|  Type definition for regFilespace parameter on dsmRegisterFS()
```

```
+--------------------------------------------------------------------------*/
typedef struct S_regFSData
{
   dsUint16_t      stVersion;                          /* structure version */
   char            *fsName;                       /* Filespace name */
   char            *fsType;                       /* Filespace type */
   dsStruct64_t    occupancy;                     /* Occupancy est. in bytes. */
   dsStruct64_t    capacity;                       /* Capacity est. in bytes. */
   dsmFSAttr       fsAttr ;                   /* platform specific attributes */
}regFSData;

#define regFSDataVersion  1

/*--------------------------------------------------------------------------+
| Type definition for dedupType used in apisessInfo                         |
+--------------------------------------------------------------------------*/
typedef enum
{
   dedupServerOnly= 0x00,             /* dedup only done on server    */
   dedupClientOrServer              /* dedup can be done on client or server */
}dsmDedupType ;

/*--------------------------------------------------------------------------+
 | Type definition for fail over configuration and status
 --------------------------------------------------------------------------*/
typedef enum
{
   failOvrNotConfigured = 0x00,
   failOvrConfigured,
   failOvrConnectedToReplServer
}dsmFailOvrCfgType ;

/*--------------------------------------------------------------------------+
|  Type definition for session info response on dsmQuerySessionInfo()       |
+--------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t     stVersion;              /* Structure version                */
      /*-----------------------------------------------------------------*/
      /*           Server information                                    */
      /*-----------------------------------------------------------------*/
   char           serverHost[DSM_MAX_SERVERNAME_LENGTH+1];
                                  /* Network host name of DSM server  */
   dsUint16_t     serverPort;             /* Server comm port on host         */
   dsmDate        serverDate;             /* Server's date/time               */
   char           serverType[DSM_MAX_SERVERTYPE_LENGTH+1];
                                 /* Server's execution platform      */
   dsUint16_t     serverVer;              /* Server's version number          */
   dsUint16_t     serverRel;              /* Server's release number          */
   dsUint16_t     serverLev;              /* Server's level number            */
   dsUint16_t     serverSubLev;           /* Server's sublevel number         */
      /*-----------------------------------------------------------------*/
      /*           Client Defaults                                       */
      /*-----------------------------------------------------------------*/
   char           nodeType[DSM_MAX_PLATFORM_LENGTH+1]; /*node/application type*/
   char           fsdelim;                /* File space delimiter             */
   char           hldelim;                /* Delimiter betw highlev & lowlev  */
   dsUint8_t      compression;            /* Compression flag                 */
   dsUint8_t      archDel;                /* Archive delete permission        */
   dsUint8_t      backDel;                /* Backup  delete permission        */
   dsUint32_t     maxBytesPerTxn;         /* for future use                   */
   dsUint16_t     maxObjPerTxn;           /* The max objects allowed in a txn */
      /*-----------------------------------------------------------------*/
      /*           Session Information                                   */
      /*-----------------------------------------------------------------*/
   char       id[DSM_MAX_ID_LENGTH+1];    /* Sign-in id node name         */
   char       owner[DSM_MAX_OWNER_LENGTH+1]; /* Sign-in owner             */
                                  /*   (for multi-user platforms)     */
   char       confFile[DSM_PATH_MAX + DSM_NAME_MAX +1];
                                  /* len is platform dep              */
```

```
                                  /* dsInit name of appl config file  */
   dsUint8_t  opNoTrace;             /* dsInit option - NoTrace = 1      */
      /*-----------------------------------------------------------------*/
      /*           Policy Data                                           */
      /*-----------------------------------------------------------------*/
   char           domainName[DSM_MAX_DOMAIN_LENGTH+1]; /* Domain name         */
   char           policySetName[DSM_MAX_PS_NAME_LENGTH+1];
                                  /* Active policy set name           */
   dsmDate        polActDate;             /* Policy set activation date       */
   char           dfltMCName[DSM_MAX_MC_NAME_LENGTH+1];/* Default Mgmt Class  */
   dsUint16_t     gpBackRetn;             /* Grace-period backup retention    */
   dsUint16_t     gpArchRetn;             /* Grace-period archive retention   */
   char           adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* adsm server name */
   dsmBool_t      archiveRetentionProtection; /* is server Retention protection enabled */
   dsStruct64_t   maxBytesPerTxn_64;         /* for future use                */
   dsmBool_t      lanFreeEnabled;            /* lan free option is set        */
   dsmDedupType   dedupType;                 /* server or clientOrServer      */
   char           accessNode[DSM_MAX_ID_LENGTH+1];    /* as node node name         */

   /*-----------------------------------------------------------------*/
   /*           Replication and fail over information                 */
   /*-----------------------------------------------------------------*/
   dsmFailOvrCfgType failOverCfgType; /* status of fail over */
   char           replServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* repl server name */
   char           homeServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* home server name */
   char           replServerHost[DSM_MAX_SERVERNAME_LENGTH+1]; /* Network host name of DSM server  */
   dsInt32_t      replServerPort;                              /* Server comm port on host         */

}ApiSessInfo;

#define ApiSessInfoVersion  6



/*-------------------------------------------------------------------------+
| Type definition for Query options response on dsmQueryCliOptions()       |
|      and dsmQuerySessOptions()                                           |
+-------------------------------------------------------------------------*/

typedef struct
{
   char        dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char        dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char        serverName[DSM_MAX_SERVERNAME_LENGTH+1];
   dsInt16_t   commMethod;
   char        serverAddress[DSM_MAX_SERVER_ADDRESS];
   char        nodeName[DSM_MAX_NODE_LENGTH+1];
   dsmBool_t   compression;
   dsmBool_t   compressalways;
   dsmBool_t   passwordAccess;
}optStruct ;

/*-------------------------------------------------------------------------+
| Type definition for LogType used in logInfo                              |
+-------------------------------------------------------------------------*/
typedef enum
{
   logServer = 0x00,             /* log msg only to server    */
   logLocal,                     /* log msg only to local error log */
   logBoth,                       /* log msg to server and to local error log */
   logNone
}dsmLogType ;

/*-------------------------------------------------------------------------+
| Type definition for logInfo parameter used on dsmLogEvent()              |
+-------------------------------------------------------------------------*/

typedef struct
{
   char        *message;   /* text of message to be logged */
```

```
   dsmLogType  logType;    /* log type : local, server, both */
}logInfo;

/*-------------------------------------------------------------------------+
| Type definition for qryRespAccessData  parameter used on dsmQueryAccess()|
+-------------------------------------------------------------------------*/

typedef struct
{
   dsUint16_t        stVersion ;                    /* structure version      */
   char              node[DSM_MAX_ID_LENGTH+1];     /* node name              */
   char              owner[DSM_MAX_OWNER_LENGTH+1]; /* owner                  */
   dsmObjName        objName ;                      /* object name            */
   dsmAccessType     accessType;                    /* archive or backup     */
   dsUint32_t        ruleNumber ;                   /* Access rule id         */
}qryRespAccessData;

#define qryRespAccessDataVersion  1

/*-------------------------------------------------------------------------+
|  Type definition for envSetUp parameter on dsmSetUp()
+-------------------------------------------------------------------------*/
typedef struct S_envSetUp
{
   dsUint16_t      stVersion;                      /* structure version */
   char            dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char            dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char            dsmiLog[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char          **argv;  /* for executables name argv[0] */
   char            logName[DSM_NAME_MAX +1];
   dsmBool_t       reserved1;         /* for future use */
   dsmBool_t       reserved2;                 /* for future use */
}envSetUp;

#define envSetUpVersion  4

/*-------------------------------------------------------------------------+
|  Type definition for dsmInitExIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmInitExIn_t
{
   dsUint16_t         stVersion;                     /* structure version */
   dsmApiVersionEx    *apiVersionExP;
   char               *clientNodeNameP;
   char               *clientOwnerNameP;
   char               *clientPasswordP;
   char               *userNameP;
   char               *userPasswordP;
   char               *applicationTypeP;
   char               *configfile;
   char               *options;
   char               dirDelimiter;
   dsmBool_t          useUnicode;
   dsmBool_t          bCrossPlatform;
   dsmBool_t          bService;
   dsmBool_t          bEncryptKeyEnabled;
   char               *encryptionPasswordP;
   dsmBool_t          useTsmBuffers;
   dsUint8_t          numTsmBuffers;
   dsmAppVersion      *appVersionP;
}dsmInitExIn_t;

#define dsmInitExInVersion 5

/*-------------------------------------------------------------------------+
|  Type definition for dsmInitExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmInitExOut_t
{
   dsUint16_t         stVersion;                   /* structure version */
```

```
   dsInt16_t          userNameAuthorities;
   dsInt16_t          infoRC;        /* error return code if encountered */
   char               adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1];
   dsUint16_t         serverVer;     /* Server's version number          */
   dsUint16_t         serverRel;     /* Server's release number          */
   dsUint16_t         serverLev;     /* Server's level number            */
   dsUint16_t         serverSubLev;  /* Server's sublevel number         */

   dsmBool_t          bIsFailOverMode; /* true if failover has occured */
   char               replServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* repl server name */
   char               homeServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* home server name */

}dsmInitExOut_t;

#define dsmInitExOutVersion 3

/*-------------------------------------------------------------------------+
| Type definition for LogType used in logInfo                              |
+-------------------------------------------------------------------------*/
typedef enum
{
   logSevInfo = 0x00,       /* information ANE4991 */
   logSevWarning,           /* warning     ANE4992 */
   logSevError,             /* Error       ANE4993 */
   logSevSevere,            /* severe      ANE4994 */
   logSevLicense,           /* License     ANE4995 */
   logSevTryBuy             /* try Buy     ANE4996 */
}dsmLogSeverity ;


/*-------------------------------------------------------------------------+
|  Type definition for dsmLogExIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmLogExIn_t
{
   dsUint16_t         stVersion;  /* structure version */
   dsmLogSeverity     severity;
   char               appMsgID[8];
   dsmLogType         logType;    /* log type : local, server, both */
   char               *message;   /* text of message to be logged */
   char               appName[DSM_MAX_PLATFORM_LENGTH];
   char               osPlatform[DSM_MAX_PLATFORM_LENGTH];
   char               appVersion[DSM_MAX_PLATFORM_LENGTH];
}dsmLogExIn_t;

#define dsmLogExInVersion 2

/*-------------------------------------------------------------------------+
|  Type definition for dsmlogExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmLogExOut_t
{
   dsUint16_t         stVersion;  /* structure version */
}dsmLogExOut_t;

#define dsmLogExOutVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmRenameIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmRenameIn_t
{
   dsUint16_t       stVersion;                    /* structure version  */
   dsUint32_t       dsmHandle;                    /* handle for session */
   dsUint8_t        repository;                   /* Backup or Archive  */
   dsmObjName       *objNameP ;                   /* object name */
   char             newHl[DSM_MAX_HL_LENGTH + 1];  /* new High level name */
   char             newLl[DSM_MAX_LL_LENGTH + 1];  /* new Low level name */
   dsmBool_t        merge;                       /* merge into existing name*/
   ObjID            objId;                        /* objId for Archive */
```

```
}dsmRenameIn_t;

#define dsmRenameInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmRenameOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmRenameOut_t
{
   dsUint16_t         stVersion;                     /* structure version */
}dsmRenameOut_t;

#define dsmRenameOutVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmEndSendObjExIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndSendObjExIn_t
{
   dsUint16_t       stVersion;                    /* structure version  */
   dsUint32_t       dsmHandle;                    /* handle for session */
}dsmEndSendObjExIn_t;

#define dsmEndSendObjExInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmEndSendObjExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndSendObjExOut_t
{
   dsUint16_t         stVersion;           /* structure version */
   dsStruct64_t       totalBytesSent;      /* total bytes read from app */
   dsmBool_t          objCompressed;       /* was object compressed */
   dsStruct64_t       totalCompressSize;   /* total size after compress */
   dsStruct64_t       totalLFBytesSent;    /* total bytes sent Lan Free */
   dsUint8_t          encryptionType;      /* type of encryption used   */
   dsmBool_t          objDeduplicated;     /* was object processed for dist. data dedup */
   dsStruct64_t       totalDedupSize;      /* total size after de-dup */
}dsmEndSendObjExOut_t;

#define dsmEndSendObjExOutVersion 3
/*-------------------------------------------------------------------------+
|  Type definition for dsmGroupHandlerIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmGroupHandlerIn_t
{
   dsUint16_t      stVersion;        /* structure version  */
   dsUint32_t      dsmHandle;        /* handle for session */
   dsUint8_t       groupType;        /* Type of group                          */
   dsUint8_t       actionType;       /* Type of group operation                */
   dsUint8_t       memberType;       /* Type of member: Leader or member       */
   dsStruct64_t    leaderObjId;      /* OBJID of the groupleader when manipulating a member */
   char            *uniqueGroupTagP; /* Unique group identifier                */
   dsmObjName      *objNameP ;       /* group leader object name */
   dsmGetList      memberObjList;    /* list of objects to remove, assign      */
}dsmGroupHandlerIn_t;

#define dsmGroupHandlerInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmGroupHandlerExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmGroupHandlerOut_t
{
   dsUint16_t         stVersion;                     /* structure version */
}dsmGroupHandlerOut_t;

#define dsmGroupHandlerOutVersion 1

/*-------------------------------------------------------------------------+
```

```
|  Type definition for dsmEndTxnExIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndTxnExIn_t
{
   dsUint16_t       stVersion;                    /* structure version  */
   dsUint32_t       dsmHandle;                    /* handle for session */
   dsUint8_t        vote;
}dsmEndTxnExIn_t;

#define dsmEndTxnExInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmEndTxnExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndTxnExOut_t
{
   dsUint16_t         stVersion;              /* structure version             */
   dsUint16_t         reason;                 /* reason code                   */
   dsStruct64_t       groupLeaderObjId;       /* groupLeader obj id returned on */
                                              /* DSM_ACTION_OPEN               */
   dsUint8_t          reserved1;              /* future use                    */
   dsUint16_t         reserved2;              /* future use                    */
}dsmEndTxnExOut_t;

#define dsmEndTxnExOutVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmEndGetDataExIn_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndGetDataExIn_t
{
   dsUint16_t     stVersion;     /* structure version  */
   dsUint32_t     dsmHandle;     /* handle for session */
}dsmEndGetDataExIn_t;

#define dsmEndGetDataExInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for dsmEndGetDataExOut_t
+-------------------------------------------------------------------------*/
typedef struct dsmEndGetDataExOut_t
{
   dsUint16_t     stVersion;         /* structure version             */
   dsUint16_t     reason;            /* reason code                   */
   dsStruct64_t   totalLFBytesRecv;  /* total lan free bytes recieved */
}dsmEndGetDataExOut_t;

#define dsmEndGetDataExOutVersion 1

/*-------------------------------------------------------------------------+
| Type definition for object list on dsmRetentionEvent()                   |
+-------------------------------------------------------------------------*/
typedef struct dsmObjList
{
   dsUint16_t       stVersion;         /* structure version               */
   dsUint32_t       numObjId;          /* number of object IDs in the list */
   ObjID            *objId;            /* list of object IDs to signal     */
}dsmObjList_t ;

#define dsmObjlistVersion 1

/*-------------------------------------------------------------------------+
| Type definition eventType used on dsmRetentionEvent                       |
+-------------------------------------------------------------------------*/
typedef enum
{
   eventRetentionActivate = 0x00,   /* signal the server that the event has occured */
   eventHoldObj,                    /* suspend delete/expire of the object          */
   eventReleaseObj                  /* Resume normal delete/expire processing       */
}dsmEventType_t;
```

```
/*-------------------------------------------------------------------------+
|  Type definition for  on dsmRetentionEvent()                             |
+-------------------------------------------------------------------------*/
typedef struct dsmRetentionEventIn_t
{
   dsUint16_t         stVersion;                  /* structure version  */
   dsUint32_t         dsmHandle;                  /* session Handle     */
   dsmEventType_t     eventType;                  /* Event type         */
   dsmObjList_t       objList;                    /* object ID          */
}dsmRetentionEventIn_t;

#define dsmRetentionEventInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for  on dsmRetentionEvent()                             |
+-------------------------------------------------------------------------*/
typedef struct dsmRetentionEventOut_t
{
   dsUint16_t       stVersion ;                  /* structure version      */
}dsmRetentionEventOut_t;

#define dsmRetentionEventOutVersion 1


/*-------------------------------------------------------------------------+
|  Type definition for  on dsmRequestBuffer()                              |
+-------------------------------------------------------------------------*/
typedef struct requestBufferIn_t
{
   dsUint16_t         stVersion;                  /* structure version  */
   dsUint32_t         dsmHandle;                  /* session Handle     */
}requestBufferIn_t;

#define requestBufferInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for  on dsmRequestBuffer()                              |
+-------------------------------------------------------------------------*/
typedef struct requestBufferOut_t
{
   dsUint16_t       stVersion ;               /* structure version */
   dsUint8_t        tsmBufferHandle;          /* handle to tsm Data buffer */
   char             *dataPtr;                 /* Address to write data to */
   dsUint32_t       bufferLen;                /* Max length of data to be written */
}requestBufferOut_t;

#define requestBufferOutVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for  on dsmReleaseBuffer()                              |
+-------------------------------------------------------------------------*/
typedef struct releaseBufferIn_t
{
   dsUint16_t         stVersion;                  /* structure version  */
   dsUint32_t         dsmHandle;                  /* session Handle     */
   dsUint8_t          tsmBufferHandle;            /* handle to tsm Data buffer */
   char               *dataPtr;                   /* Address to write data to */
}releaseBufferIn_t;

#define releaseBufferInVersion 1

/*-------------------------------------------------------------------------+
|  Type definition for  on dsmReleaseBuffer()                              |
+-------------------------------------------------------------------------*/
typedef struct releaseBufferOut_t
{
   dsUint16_t       stVersion ;                 /* structure version */
}releaseBufferOut_t;
```

```
#define releaseBufferOutVersion 1

/*--------------------------------------------------------------------------+
|  Type definition for  on dsmGetBufferData()                               |
+--------------------------------------------------------------------------*/
typedef struct getBufferDataIn_t
{
   dsUint16_t         stVersion;                  /* structure version  */
   dsUint32_t         dsmHandle;                  /* session Handle     */
}getBufferDataIn_t;

#define getBufferDataInVersion 1

/*--------------------------------------------------------------------------+
|  Type definition for  on dsmGetBufferData()                                |
+--------------------------------------------------------------------------*/
typedef struct getBufferDataOut_t
{
   dsUint16_t       stVersion ;            /* structure version */
   dsUint8_t        tsmBufferHandle;       /* handle to tsm Data buffer */
   char             *dataPtr;              /* Address of actual data to read */
   dsUint32_t       numBytes;              /* Actual number of bytes to read from dataPtr*/
}getBufferDataOut_t;

#define getBufferDataOutVersion 1

/*--------------------------------------------------------------------------+
|  Type definition for  on dsmSendBufferData()                              |
+--------------------------------------------------------------------------*/
typedef struct sendBufferDataIn_t
{
   dsUint16_t         stVersion;             /* structure version  */
   dsUint32_t         dsmHandle;             /* session Handle     */
   dsUint8_t          tsmBufferHandle;       /* handle to tsm Data buffer */
   char               *dataPtr;              /* Address of actual data to send */
   dsUint32_t         numBytes;              /* Actual number of bytes to send from dataPtr*/
}sendBufferDataIn_t;

#define sendBufferDataInVersion 1

/*--------------------------------------------------------------------------+
|  Type definition for  on dsmSendBufferData()                              |
+--------------------------------------------------------------------------*/
typedef struct sendBufferDataOut_t
{
   dsUint16_t       stVersion ;            /* structure version */
}sendBufferDataOut_t;

#define sendBufferDataOutVersion 1

/*--------------------------------------------------------------------------+
|  Type definition for dsmUpdateObjExIn_t
+--------------------------------------------------------------------------*/
typedef struct dsmUpdateObjExIn_t
{
   dsUint16_t      stVersion;            /* structure version */
   dsUint32_t      dsmHandle;            /* session Handle      */
   dsmSendType     sendType;             /* send type back/arch */
   char            *descrP;              /* archive description */
   dsmObjName      *objNameP;            /* objName             */
   ObjAttr         *objAttrPtr;          /* attribute           */
   dsUint32_t      objUpdAct;            /* update action       */
   ObjID           archObjId;            /* objId for archive   */
}dsmUpdateObjExIn_t;

#define dsmUpdateObjExInVersion 1

/*--------------------------------------------------------------------------+
|  Type definition for dsmUpdateObjExOut_t
+--------------------------------------------------------------------------*/
```

```
typedef struct dsmUpdateObjExOut_t
{
    dsUint16_t         stVersion;        /* structure version */
}dsmUpdateObjExOut_t;

#define dsmUpdateObjExOutVersion 1

#if (_OPSYS_TYPE == DS_WINNT) && !defined(_WIN64)
#pragma pack()
#endif


#ifdef _MAC
#pragma options align=reset
#endif
#endif /* _H_DSMAPITD */

/*****************************************************************
 * Tivoli Storage Manager                                        *
 * API Client Component                                          *
 *                                                               *
 * (C) Copyright IBM Corporation 1993,2010                      *
 *****************************************************************/

/*****************************************************************************
 * Header File Name: tsmapitd.h
 *
 * Environment:     ***********************************************
 *                  ** This is a platform-independent source file **
 *
 *                  ***********************************************
 *
 * Design Notes:    This file contains basic data types and constants
 *                  includable by all client source files. The constants
 *                  within this file should be set properly for the
 *                  particular machine and operating system on which the
 *                  client software is to be run.
 *
 *                  Platform specific definitions are included in dsmapips.h
 *
 * Descriptive-name: Definitions for Tivoli Storage manager API constants
 *---------------------------------------------------------------------------*/

#ifndef _H_TSMAPITD
#define _H_TSMAPITD


/*=== set the structure alignment to pack the structures ===*/
#if _OPSYS_TYPE == DS_WINNT
#ifdef _WIN64
#pragma pack(8)
#else
#pragma pack(1)
#endif
#endif

#ifdef _MAC
#pragma options align = packed
#endif

/*================================================================
 Win32 applications using the tsm interface must use the
 -DUNICODE flag during compilation.
 ================================================================*/
#if _OPSYS_TYPE == DS_WINNT && !defined(DSMAPILIB)
#ifndef UNICODE
#error "Win32 applications using the TSM interface MUST be compiled with the -DUNICODE flag"
```

```
#endif
#endif

/*============================================================
 Mac OS X applications using the tsm interface must use the
 -DUNICODE flag during compilation.
 ============================================================*/
#if _OPSYS_TYPE == DS_MACOS && !defined(DSMAPILIB)
#ifndef UNICODE
#error "Mac OS X applications using the TSM interface MUST be compiled with the -DUNICODE flag"
#endif
#endif

/*-------------------------------------------------------------------------+
 | Type definition for dsmGetType parameter on tsmBeginGetData()           |
 +-------------------------------------------------------------------------*/
typedef enum
{
   gtTsmBackup = 0x00,                    /* Backup processing type    */
   gtTsmArchive                           /* Archive processing type   */
} tsmGetType ;

/*-------------------------------------------------------------------------+
 |  Type definition for dsmQueryType parameter on tsmBeginQuery()          |
 +-------------------------------------------------------------------------*/
typedef enum
{
   qtTsmArchive = 0x00,                    /* Archive query type        */
   qtTsmBackup,                            /* Backup query type         */
   qtTsmBackupActive,                      /* Fast query for active backup files */
   qtTsmFilespace,                         /* Filespace query type      */
   qtTsmMC,                                /* Mgmt. class query type    */
    qtTsmReserved1,                         /* future use                */
    qtTsmReserved2,                         /* future use                */
    qtTsmReserved3,                         /* future use                */
    qtTsmReserved4,                         /* future use                */
   qtTsmBackupGroups,                      /* All group leaders in a specific filespace */
   qtTsmOpenGroups,                        /* All group members associated with a leader */
    qtTsmReserved5,                         /* future use                */
   qtTsmProxyNodeAuth,                     /* nodes that this node can proxy to   */
    qtTsmProxyNodePeer,                     /* peer nodes under this target node  */
    qtTsmReserved6,                         /* future use                */
    qtTsmReserved7,                         /* future use                */
    qtTsmReserved8                          /* future use                */
} tsmQueryType ;

/*-------------------------------------------------------------------------+
 |  Type definition sendType parameter on tsmBindMC() and tsmSendObj()     |
 +-------------------------------------------------------------------------*/
typedef enum
{
   stTsmBackup = 0x00,                        /* Backup processing type    */
   stTsmArchive,                              /* Archive processing type   */
   stTsmBackupMountWait,           /* Backup processing with mountwait on   */
   stTsmArchiveMountWait           /* Archive processing with mountwait on  */
} tsmSendType ;

/*-------------------------------------------------------------------------+
 |  Type definition for delType parameter on tsmDeleteObj()                |
 +-------------------------------------------------------------------------*/
typedef enum
{
   dtTsmArchive = 0x00,                             /* Archive delete type */
   dtTsmBackup,                           /* Backup delete (deactivate) type */
   dtTsmBackupID                         /* Backup delete (remove)     type */
} tsmDelType ;
```

```
/*--------------------------------------------------------------------+
 |  Type definition sendType parameter on tsmSetAccess()              |
 +--------------------------------------------------------------------*/
typedef enum
{
   atTsmBackup = 0x00,                        /* Backup processing type    */
   atTsmArchive                              /* Archive processing type   */
}tsmAccessType;

/*--------------------------------------------------------------------+
 |  Type definition for Overwrite  parameter on tsmSendObj()          |
 +--------------------------------------------------------------------*/
typedef enum
{
   owIGNORE = 0x00,
   owYES,
   owNO
}tsmOwType;


/*--------------------------------------------------------------------+
 |  Type definition for API Version on tsmInit() and tsmQueryApiVersion()   |
 +--------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t stVersion;     /* Structure version               */
   dsUint16_t version;       /* API version                     */
   dsUint16_t release;       /* API release                     */
   dsUint16_t level;         /* API level                       */
   dsUint16_t subLevel;      /* API sub level                   */
   dsmBool_t  unicode;       /* API unicode?                    */
} tsmApiVersionEx;

#define tsmApiVersionExVer       2

/*--------------------------------------------------------------------+
|  Type definition for Application Version on tsmInit()               |
+--------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t   stVersion;            /* Structure version           */
   dsUint16_t   applicationVersion;   /* application version number   */
   dsUint16_t   applicationRelease;   /* application release number   */
   dsUint16_t   applicationLevel;     /* application level number     */
   dsUint16_t   applicationSubLevel;  /* application sub level number */
} tsmAppVersion;

#define tsmAppVersionVer      1


/*--------------------------------------------------------------------+
 |  Type definition for object name used on BindMC, Send, Delete, Query     |
 +--------------------------------------------------------------------*/

typedef struct tsmObjName
{
   dsChar_t   fs[DSM_MAX_FSNAME_LENGTH + 1] ;              /* Filespace name */
   dsChar_t   hl[DSM_MAX_HL_LENGTH + 1] ;                 /* High level name */
   dsChar_t   ll[DSM_MAX_LL_LENGTH + 1] ;                  /* Low level name */
   dsUint8_t  objType;          /* for object type values, see defines above */
   dsChar_t   dirDelimiter;
} tsmObjName;

/*--------------------------------------------------------------------+
 |  Type definition for Backup delete info on dsmDeleteObj()          |
 +--------------------------------------------------------------------*/
typedef struct tsmDelBack
```

```
{
   dsUint16_t       stVersion ;                   /* structure version      */
   tsmObjName       *objNameP ;                   /* object name            */
   dsUint32_t       copyGroup ;                   /* copy group             */
} tsmDelBack ;

#define  tsmDelBackVersion    1

/*-------------------------------------------------------------------------+
 |  Type definition for Archive delete info on dsmDeleteObj()              |
 +-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                   /* structure version      */
   dsStruct64_t     objId ;                       /* object ID              */
} tsmDelArch ;

#define  tsmDelArchVersion  1

/*-------------------------------------------------------------------------+
 |  Type definition for Backup ID delete info on dsmDeleteObj()            |
 +-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t       stVersion ;                   /* structure version      */
   dsStruct64_t       objId ;                       /* object ID              */
} tsmDelBackID;

#define  tsmDelBackIDVersion  1


/*-------------------------------------------------------------------------+
 |  Type definition for delete info on dsmDeleteObj()                      |
 +-------------------------------------------------------------------------*/
typedef union
{
   tsmDelBack   backInfo ;
   tsmDelArch   archInfo ;
   tsmDelBackID backIDInfo;
} tsmDelInfo ;

/*-------------------------------------------------------------------------+
 |  Type definition for Object Attribute parameter on dsmSendObj()         |
 +-------------------------------------------------------------------------*/
typedef struct tsmObjAttr
{
   dsUint16_t   stVersion;                /* Structure version               */
   dsChar_t     owner[DSM_MAX_OWNER_LENGTH + 1];      /* object owner         */
   dsStruct64_t sizeEstimate;             /* Size estimate in bytes of the object */
   dsmBool_t    objCompressed;            /* Is object already compressed?    */
   dsUint16_t   objInfoLength;            /* length of object-dependent info  */
   char         *objInfo;                 /* object-dependent info byte buffer */
   dsChar_t     *mcNameP;                 /* mgmnt class name for override    */
   tsmOwType    reserved1;                /* for future use                   */
   tsmOwType    reserved2;                /* for future use                   */
   dsmBool_t    disableDeduplication;     /* force no dedup for this object */
   dsmBool_t    useExtObjInfo;             /* use ext objinfo up to 1536       */
} tsmObjAttr;

#define tsmObjAttrVersion 5

/*-------------------------------------------------------------------------+
 | Type definition for mcBindKey returned on dsmBindMC()                   |
 +-------------------------------------------------------------------------*/
typedef struct tsmMcBindKey
{
   dsUint16_t  stVersion;                   /* structure version         */
```

```
   dsChar_t    mcName[DSM_MAX_MC_NAME_LENGTH + 1];
   /* Name of mc bound to object. */
   dsmBool_t   backup_cg_exists;                          /* True/false */
   dsmBool_t   archive_cg_exists;                         /* True/false */
   dsChar_t    backup_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
   /* Backup copy dest. name */
   dsChar_t    archive_copy_dest[DSM_MAX_CG_DEST_LENGTH + 1];
   /* Arch copy dest.name */
} tsmMcBindKey;

#define tsmMcBindKeyVersion  1


/*-------------------------------------------------------------------------+
 |  Type definition for Mgmt Class queryBuffer on dsmBeginQuery()          |
 +-------------------------------------------------------------------------*/
typedef struct tsmQryMCData
{
   dsUint16_t   stVersion;                            /* structure version */
   dsChar_t     *mcName;                      /* Mgmt class name */
   /* single name to get one or empty string to get all*/
   dsmBool_t    mcDetail;                            /* Want details or not? */
} tsmQryMCData;

#define tsmQryMCDataVersion  1


/*-------------------------------------------------------------------------+
 |  Type definition for Archive Copy Group details on Query MC response    |
 +-------------------------------------------------------------------------*/
typedef struct tsmArchDetailCG
{
   dsChar_t     cgName[DSM_MAX_CG_NAME_LENGTH + 1];     /* Copy group name */
   dsUint16_t   frequency;                     /* Copy (archive) frequency */
   dsUint16_t   retainVers;                             /* Retain version */
   dsUint8_t    copySer;     /* for copy serialization values, see defines */
   dsUint8_t    copyMode;       /* for copy mode values, see defines above */
   dsChar_t     destName[DSM_MAX_CG_DEST_LENGTH + 1];    /* Copy dest name */
   dsmBool_t    bLanFreeDest;         /* Destination has lan free path?    */
   dsmBool_t    reserved;                              /* Not currently used */
   dsUint8_t    retainInit;             /* possible values see dsmapitd.h  */
   dsUint16_t   retainMin;              /* if retInit is EVENT num of days */
   dsmBool_t    bDeduplicate;             /* destination has dedup enabled     */
}tsmArchDetailCG;

/*-------------------------------------------------------------------------+
 |  Type definition for Backup Copy Group details on Query MC response     |
 +-------------------------------------------------------------------------*/
typedef struct tsmBackupDetailCG
{
   dsChar_t     cgName[DSM_MAX_CG_NAME_LENGTH + 1];       /* Copy group name */
   dsUint16_t   frequency;                               /* Backup frequency */
   dsUint16_t   verDataExst;                           /* Versions data exists */
   dsUint16_t   verDataDltd;                          /* Versions data deleted */
   dsUint16_t   retXtraVers;                          /* Retain extra versions */
   dsUint16_t   retOnlyVers;                           /* Retain only versions */
   dsUint8_t    copySer;       /* for copy serialization values, see defines */
   dsUint8_t    copyMode;         /* for copy mode values, see defines above */
   dsChar_t     destName[DSM_MAX_CG_DEST_LENGTH + 1];      /* Copy dest name */
   dsmBool_t    bLanFreeDest;           /* Destination has lan free path?    */
   dsmBool_t    reserved;                              /* Not currently used */
   dsmBool_t    bDeduplicate;             /* destination has dedup enabled     */
}tsmBackupDetailCG;


/*-------------------------------------------------------------------------+
 |  Type definition for Query Mgmt Class detail response on dsmGetNextQObj()|
```

```
 +-------------------------------------------------------------------------*/
typedef struct tsmQryRespMCDetailData
{
   dsUint16_t      stVersion;                       /* structure version */
   dsChar_t        mcName[DSM_MAX_MC_NAME_LENGTH + 1];        /* mc name */
   dsChar_t        mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1]; /*mc description */
   archDetailCG    archDet;                /* Archive copy group detail */
   backupDetailCG  backupDet;               /* Backup copy group detail */
} tsmQryRespMCDetailData;

#define tsmQryRespMCDetailDataVersion  4

/*-------------------------------------------------------------------------+
 | Type definition for Query Mgmt Class summary response on dsmGetNextQObj()|
 +-------------------------------------------------------------------------*/
typedef struct tsmQryRespMCData
{
   dsUint16_t   stVersion;                             /* structure version */
   dsChar_t     mcName[DSM_MAX_MC_NAME_LENGTH + 1];              /* mc name */
   dsChar_t     mcDesc[DSM_MAX_MC_DESCR_LENGTH + 1];       /* mc description */
}tsmQryRespMCData;

#define tsmQryRespMCDataVersion  1

/*-------------------------------------------------------------------------+
 |  Type definition for Archive queryBuffer on tsmBeginQuery()              |
 +-------------------------------------------------------------------------*/
typedef struct tsmQryArchiveData
{
   dsUint16_t   stVersion;                         /* structure version */
   tsmObjName   *objName;                    /* Full dsm name of object */
   dsChar_t     *owner;                              /* owner name */
   /* for maximum date boundaries, see defines above */
   dsmDate      insDateLowerBound;      /* low bound archive insert date */
   dsmDate      insDateUpperBound;       /* hi bound archive insert date */
   dsmDate      expDateLowerBound;          /* low bound expiration date */
   dsmDate      expDateUpperBound;           /* hi bound expiration date */
   dsChar_t     *descr;                /* archive description */
} tsmQryArchiveData;

#define tsmQryArchiveDataVersion  1

/*-------------------------------------------------------------------------+
| Type definition for Query Archive response on dsmGetNextQObj()           |
+-------------------------------------------------------------------------*/
typedef struct tsmQryRespArchiveData
{
    dsUint16_t      stVersion;                          /* structure version */
    tsmObjName      objName;                      /* Filespace name qualifier */
    dsUint32_t      copyGroup;                          /* copy group number */
    dsChar_t        mcName[DSM_MAX_MC_NAME_LENGTH + 1];            /* mc name */
    dsChar_t        owner[DSM_MAX_OWNER_LENGTH + 1];           /* owner name */
    dsStruct64_t    objId;                                  /* Unique copy id */
    dsStruct64_t    reserved;                       /* backward compatability */
    dsUint8_t       mediaClass;                         /* media access class */
    dsmDate         insDate;                       /* archive insertion date */
    dsmDate         expDate;                    /* expiration date for object */
    dsChar_t        descr[DSM_MAX_DESCR_LENGTH + 1];   /* archive description */
    dsUint16_t      objInfolen;             /* length of object-dependent info*/
    dsUint8_t       reservedObjInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
    dsUint160_t     restoreOrderExt;                        /* restore order */
    dsStruct64_t    sizeEstimate;              /* size estimate stored by user*/
    dsUint8_t       compressType;                         /* Compression flag */
    dsUint8_t       retentionInitiated; /* object waiting on retention event  */
    dsUint8_t       objHeld; /* object is on "hold" see dsmapitd.h for values */
    dsUint8_t       encryptionType;                     /* type of encryption */
    dsmBool_t       clientDeduplicated;           /* obj deduplicated by API*/
```

```
    dsUint8_t       objInfo[DSM_MAX_EXT_OBJINFO_LENGTH];  /*object-dependent info */
    dsChar_t        compressAlg[DSM_MAX_COMPRESSTYPE_LENGTH + 1];  /* compression algorithm name */
} tsmQryRespArchiveData;

#define tsmQryRespArchiveDataVersion  7

/*------------------------------------------------------------------------+
 |  Type definition for Archive sendBuff parameter on dsmSendObj()        |
 +------------------------------------------------------------------------*/
typedef struct tsmSndArchiveData
{
   dsUint16_t   stVersion;                  /* structure version */
   dsChar_t     *descr;                   /* archive description */
} tsmSndArchiveData;

#define tsmSndArchiveDataVersion  1

/*------------------------------------------------------------------------+
 |  Type definition for Backup queryBuffer on dsmBeginQuery()             |
 +------------------------------------------------------------------------*/
typedef struct tsmQryBackupData
{
   dsUint16_t   stVersion;         /* structure version */
   tsmObjName   *objName;          /* full dsm name of object */
   dsChar_t     *owner;            /* owner name */
   dsUint8_t    objState;          /* object state selector */
   dsmDate      pitDate;           /* Date value for point in time restore */
   /* for possible values, see defines above */
    dsUint32_t  reserved1;
    dsUint32_t  reserved2;
} tsmQryBackupData;

#define tsmQryBackupDataVersion  3

/*------------------------------------------------------------------------+
 | Type definition for Query Backup response on dsmGetNextQObj()          |
 +------------------------------------------------------------------------*/
typedef struct tsmQryRespBackupData
{
   dsUint16_t      stVersion;                          /* structure version */
   tsmObjName      objName;                       /* full dsm name of object */
   dsUint32_t      copyGroup;                          /* copy group number */
   dsChar_t        mcName[DSM_MAX_MC_NAME_LENGTH + 1];           /* mc name */
   dsChar_t        owner[DSM_MAX_OWNER_LENGTH + 1];          /* owner name */
   dsStruct64_t    objId;                               /* Unique object id */
   dsStruct64_t    reserved;                       /* backward compatability */
   dsUint8_t       mediaClass;                        /* media access class */
   dsUint8_t       objState;                      /* Obj state, active, etc. */
   dsmDate         insDate;                         /* backup insertion date */
   dsmDate         expDate;                    /* expiration date for object */
   dsUint16_t      objInfolen;             /* length of object-dependent info*/
   dsUint8_t       reservedObjInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
   dsUint160_t     restoreOrderExt;                       /* restore order */
   dsStruct64_t    sizeEstimate;             /* size estimate stored by user */
   dsStruct64_t    baseObjId;
   dsUint16_t      baseObjInfolen;             /* length of base object-dependent info*/
   dsUint8_t       baseObjInfo[DSM_MAX_OBJINFO_LENGTH];  /* base object-dependent info */
   dsUint160_t     baseRestoreOrder;                       /* restore order */
   dsUint32_t      fsID;
   dsUint8_t       compressType;
   dsmBool_t       isGroupLeader;
   dsmBool_t       isOpenGroup;
   dsUint8_t       reserved1;               /* for future use */
   dsmBool_t       reserved2;               /* for future use */
   dsUint16_t      reserved3;               /* for future use */
   reservedInfo_t  *reserved4;              /* for future use */
   dsUint8_t       encryptionType;          /* type of encryption */
```

```
   dsmBool_t       clientDeduplicated;     /* obj deduplicated by API*/
   dsUint8_t       objInfo[DSM_MAX_EXT_OBJINFO_LENGTH];  /*object-dependent info */
   dsChar_t        compressAlg[DSM_MAX_COMPRESSTYPE_LENGTH + 1];  /* compression algorithm name */
} tsmQryRespBackupData;

#define tsmQryRespBackupDataVersion  8
/*-------------------------------------------------------------------------+
 |  Type definition for Active Backup queryBuffer on dsmBeginQuery()
 |
 |  Notes:  For the active backup query, only the fs (filespace) and objType
 |          fields of objName need be set.  objType can only be set to
 |          DSM_OBJ_FILE or DSM_OBJ_DIRECTORY.  DSM_OBJ_ANY_TYPE will not
 |          find a match on the query.
 +-------------------------------------------------------------------------*/
typedef struct tsmQryABackupData
{
   dsUint16_t      stVersion;                          /* structure version */
   tsmObjName      *objName;                     /* Only fs and objtype used */
} tsmQryABackupData;

#define tsmQryABackupDataVersion  1

/*-------------------------------------------------------------------------+
 | Type definition for Query Active Backup response on dsmGetNextQObj()    |
 +-------------------------------------------------------------------------*/
typedef struct tsmQryARespBackupData
{
   dsUint16_t  stVersion;                          /* structure version */
   tsmObjName  objName;                       /* full dsm name of object */
   dsUint32_t  copyGroup;                          /* copy group number */
   dsChar_t    mcName[DSM_MAX_MC_NAME_LENGTH + 1];/*management class name*/
   dsChar_t    owner[DSM_MAX_OWNER_LENGTH + 1];            /* owner name */
   dsmDate     insDate;                         /* backup insertion date */
   dsUint16_t  objInfolen;             /* length of object-dependent info*/
   dsUint8_t   reservedObjInfo[DSM_MAX_OBJINFO_LENGTH];  /*object-dependent info */
   dsUint8_t  objInfo[DSM_MAX_EXT_OBJINFO_LENGTH];  /*object-dependent info */
} tsmQryARespBackupData;

#define tsmQryARespBackupDataVersion  2

/*-------------------------------------------------------------------------+
 |  Type definition for Backup queryBuffer on dsmBeginQuery()               |
 +-------------------------------------------------------------------------*/
typedef struct tsmQryBackupGroups
{
   dsUint16_t   stVersion;         /* structure version */
   dsUint8_t    groupType;
   dsChar_t     *fsName;
   dsChar_t     *owner;
   dsStruct64_t groupLeaderObjId;
   dsUint8_t    objType;
   dsUint32_t   reserved1;
   dsUint32_t   reserverd2;
   dsmBool_t    noRestoreOrder;
   dsmBool_t    noGroupInfo;
   dsChar_t     *hl;
} tsmQryBackupGroups;

#define tsmQryBackupGroupsVersion 4

/*-------------------------------------------------------------------------+
 |  Type definition for proxynode queryBuffer on tsmBeginQuery()            |
 +-------------------------------------------------------------------------*/
typedef struct tsmQryProxyNodeData
{
   dsUint16_t  stVersion;                   /* structure version */
   dsChar_t    *targetNodeName;             /* target node name      */
```

```
}tsmQryProxyNodeData;

#define tsmQryProxyNodeDataVersion  1

/*-------------------------------------------------------------------------+
 | Type definition for qryRespProxyNodeData  parameter used on tsmGetNextQObj()|
 +-------------------------------------------------------------------------*/

typedef struct tsmQryRespProxyNodeData
{
   dsUint16_t        stVersion ;                          /* structure version */
   dsChar_t          targetNodeName[DSM_MAX_ID_LENGTH+1];  /* target node name  */
   dsChar_t          peerNodeName[DSM_MAX_ID_LENGTH+1];    /* peer node name    */
   dsChar_t          hlAddress[DSM_MAX_ID_LENGTH+1];       /* peer hlAddress    */
   dsChar_t          llAddress[DSM_MAX_ID_LENGTH+1];       /* peer llAddress    */
}tsmQryRespProxyNodeData;

#define tsmQryRespProxyNodeDataVersion  1

/*-------------------------------------------------------------------------+
 |  Type definition for WINNT and OS/2 Filespace attributes                |
 +-------------------------------------------------------------------------*/
typedef struct tsmDosFSAttrib
{
   osChar_t     driveLetter ;          /* drive letter for filespace    */
   dsUint16_t   fsInfoLength;          /* fsInfo length used            */
   osChar_t     fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
} tsmDosFSAttrib ;

/*-------------------------------------------------------------------------+
 |  Type definition for UNIX Filespace attributes                          |
 +-------------------------------------------------------------------------*/
typedef struct tsmUnixFSAttrib
{
   dsUint16_t   fsInfoLength;          /* fsInfo length used            */
   osChar_t     fsInfo[DSM_MAX_FSINFO_LENGTH];/*caller-determined data  */
} tsmUnixFSAttrib ;

/*-------------------------------------------------------------------------+
 |  Type definition for NetWare Filespace attributes                       |
 +-------------------------------------------------------------------------*/
typedef tsmUnixFSAttrib tsmNetwareFSAttrib;

/*-------------------------------------------------------------------------+
 |  Type definition for Filespace attributes on all Filespace calls        |
 +-------------------------------------------------------------------------*/
typedef union
{
   tsmNetwareFSAttrib  netwareFSAttr;
   tsmUnixFSAttrib     unixFSAttr ;
   tsmDosFSAttrib      dosFSAttr ;
} tsmFSAttr ;

/*-------------------------------------------------------------------------+
 |  Type definition for fsUpd parameter on dsmUpdateFS()
 +-------------------------------------------------------------------------*/
typedef struct   tsmFSUpd
{
   dsUint16_t      stVersion ;             /* structure version             */
   dsChar_t        *fsType ;               /* filespace type                */
   dsStruct64_t    occupancy ;             /* occupancy estimate            */
   dsStruct64_t    capacity  ;             /* capacity estimate             */
   tsmFSAttr       fsAttr ;                /* platform specific attributes  */
} tsmFSUpd ;

#define tsmFSUpdVersion  1
```

```
/*-------------------------------------------------------------------------+
 |  Type definition for Filespace queryBuffer on dsmBeginQuery()           |
 +-------------------------------------------------------------------------*/
typedef struct tsmQryFSData
{
   dsUint16_t  stVersion;                /* structure version */
   dsChar_t    *fsName;                  /* File space name */
} tsmQryFSData;

#define tsmQryFSDataVersion  1

/*-------------------------------------------------------------------------+
 | Type definition for Query Filespace response on dsmGetNextQObj()        |
 +-------------------------------------------------------------------------*/
typedef struct tsmQryRespFSData
{
   dsUint16_t     stVersion;                         /* structure version          */
   dsChar_t       fsName[DSM_MAX_FSNAME_LENGTH + 1];  /* Filespace name             */
   dsChar_t       fsType[DSM_MAX_FSTYPE_LENGTH + 1] ; /* Filespace type             */
   dsStruct64_t   occupancy;                /* Occupancy est. in bytes.           */
   dsStruct64_t   capacity;                 /* Capacity est. in bytes.            */
   tsmFSAttr      fsAttr ;                  /* platform specific attributes       */
   dsmDate        backStartDate;            /* start backup date                  */
   dsmDate        backCompleteDate;         /* end backup Date                    */
    dsmDate        reserved1    ;           /* For future use                     */
   dsmBool_t      bIsUnicode;
   dsUint32_t     fsID;
   dsmDate        lastReplStartDate;        /* The last time replication was started */
   dsmDate        lastReplCmpltDate;        /* The last time replication completed   */
                                            /*   (could have had a failure,          */
                                            /*    but it still completes)            */
   dsmDate        lastBackOpDateFromServer; /* The last store time stamp the client  */
                                            /*    saved on the server               */
   dsmDate        lastArchOpDateFromServer; /* The last store time stamp the client  */
                                            /*    saved on the server               */
   dsmDate        lastSpMgOpDateFromServer; /* The last store time stamp the client  */
                                            /*    saved on the server               */
   dsmDate        lastBackOpDateFromLocal;  /* The last store time stamp the client  */
                                            /*    saved on the Local                */
   dsmDate        lastArchOpDateFromLocal;  /* The last store time stamp the client  */
                                            /*    saved on the Local                */
   dsmDate        lastSpMgOpDateFromLocal;  /* The last store time stamp the client  */
                                            /*    saved on the Local                */
   dsInt32_t      failOverWriteDelay;       /* Minutes for client to wait before  allowed  */
                                            /* to store to this Repl srvr, Specail codes:  */
                                            /* NO_ACCESS(-1), ACCESS_RDONLY (-2)           */
} tsmQryRespFSData;

#define tsmQryRespFSDataVersion 5

/*-------------------------------------------------------------------------+
 |  Type definition for regFilespace parameter on dsmRegisterFS()
 +-------------------------------------------------------------------------*/
typedef struct tsmRegFSData
{
   dsUint16_t      stVersion;                    /* structure version */
   dsChar_t        *fsName;                      /* Filespace name */
   dsChar_t        *fsType;                      /* Filespace type */
   dsStruct64_t    occupancy;                 /* Occupancy est. in bytes. */
   dsStruct64_t    capacity;                   /* Capacity est. in bytes. */
   tsmFSAttr       fsAttr ;                /* platform specific attributes */
} tsmRegFSData;

#define tsmRegFSDataVersion  1

/*-------------------------------------------------------------------------+
 |  Type definition for session info response on dsmQuerySessionInfo()     |
```

```
 +-------------------------------------------------------------------------*/
typedef struct
{
   dsUint16_t     stVersion;           /* Structure version                */
   /*-----------------------------------------------------------------*/
   /*           Server information                                    */
   /*-----------------------------------------------------------------*/
   dsChar_t       serverHost[DSM_MAX_SERVERNAME_LENGTH+1];
   /* Network host name of DSM server  */
   dsUint16_t     serverPort;             /* Server comm port on host         */
   dsmDate        serverDate;             /* Server's date/time               */
   dsChar_t       serverType[DSM_MAX_SERVERTYPE_LENGTH+1];
   /* Server's execution platform      */
   dsUint16_t     serverVer;              /* Server's version number          */
   dsUint16_t     serverRel;              /* Server's release number          */
   dsUint16_t     serverLev;              /* Server's level number            */
   dsUint16_t     serverSubLev;           /* Server's sublevel number         */
   /*-----------------------------------------------------------------*/
   /*           Client Defaults                                       */
   /*-----------------------------------------------------------------*/
   dsChar_t       nodeType[DSM_MAX_PLATFORM_LENGTH+1]; /*node/application type*/
   dsChar_t       fsdelim;                /* File space delimiter             */
   dsChar_t       hldelim;                /* Delimiter betw highlev & lowlev  */
   dsUint8_t      compression;            /* Compression flag                 */
   dsUint8_t      archDel;                /* Archive delete permission        */
   dsUint8_t      backDel;                /* Backup  delete permission        */
   dsUint32_t     maxBytesPerTxn;         /* for future use                   */
   dsUint16_t     maxObjPerTxn;           /* The max objects allowed in a txn */
   /*-----------------------------------------------------------------*/
   /*           Session Information                                   */
   /*-----------------------------------------------------------------*/
   dsChar_t       id[DSM_MAX_ID_LENGTH+1];    /* Sign-in id node name         */
   dsChar_t       owner[DSM_MAX_OWNER_LENGTH+1]; /* Sign-in owner             */
   /*   (for multi-user platforms)     */
   dsChar_t       confFile[DSM_PATH_MAX + DSM_NAME_MAX +1];
   /* len is platform dep              */
   /* dsInit name of appl config file  */
   dsUint8_t      opNoTrace;              /* dsInit option - NoTrace = 1      */
   /*-----------------------------------------------------------------*/
   /*           Policy Data                                           */
   /*-----------------------------------------------------------------*/
   dsChar_t       domainName[DSM_MAX_DOMAIN_LENGTH+1]; /* Domain name         */
   dsChar_t       policySetName[DSM_MAX_PS_NAME_LENGTH+1];
   /* Active policy set name           */
   dsmDate        polActDate;             /* Policy set activation date       */
   dsChar_t       dfltMCName[DSM_MAX_MC_NAME_LENGTH+1];/* Default Mgmt Class  */
   dsUint16_t     gpBackRetn;             /* Grace-period backup retention    */
   dsUint16_t     gpArchRetn;             /* Grace-period archive retention   */
   dsChar_t       adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* adsm server name */
  dsmBool_t      archiveRetentionProtection; /* is server Retention protection enabled */
   dsUint64_t     maxBytesPerTxn_64;         /* for future use                  */
   dsmBool_t      lanFreeEnabled;            /* lan free option is set         */
   dsmDedupType   dedupType;                 /* server or clientOrServer       */
   dsChar_t       accessNode[DSM_MAX_ID_LENGTH+1];    /* as node node name         */

   /*-----------------------------------------------------------------*/
   /*           Replication and fail over information                 */
   /*-----------------------------------------------------------------*/
   dsmFailOvrCfgType failOverCfgType; /* status of fail over */
   dsChar_t       replServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* repl server name */
   dsChar_t       homeServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* home server name */
   dsChar_t       replServerHost[DSM_MAX_SERVERNAME_LENGTH+1]; /* Network host name of DSM server  */
   dsInt32_t      replServerPort;                              /* Server comm port on host         */

} tsmApiSessInfo;

#define tsmApiSessInfoVersion  6
```

```
/*-------------------------------------------------------------------------+
 | Type definition for Query options response on dsmQueryCliOptions()      |
 |      and dsmQuerySessOptions()                                          |
 +-------------------------------------------------------------------------*/

typedef struct
{
   dsUint16_t  stVersion;
   dsChar_t    dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
   dsChar_t    dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
   dsChar_t    serverName[DSM_MAX_SERVERNAME_LENGTH+1];
   dsInt16_t   commMethod;
   dsChar_t    serverAddress[DSM_MAX_SERVER_ADDRESS];
   dsChar_t    nodeName[DSM_MAX_NODE_LENGTH+1];
   dsmBool_t   compression;
   dsmBool_t   compressalways;
   dsmBool_t   passwordAccess;
}tsmOptStruct ;

#define tsmOptStructVersion  1

/*-------------------------------------------------------------------------+
 | Type definition for qryRespAccessData  parameter used on dsmQueryAccess()|
 +-------------------------------------------------------------------------*/

typedef struct
{
   dsUint16_t        stVersion ;                   /* structure version      */
   dsChar_t          node[DSM_MAX_ID_LENGTH+1];    /* node name              */
   dsChar_t          owner[DSM_MAX_OWNER_LENGTH+1]; /* owner                 */
   tsmObjName        objName ;                     /* object name            */
   dsmAccessType     accessType;                   /* archive or backup     */
   dsUint32_t        ruleNumber ;                  /* Access rule id         */

}tsmQryRespAccessData;

#define tsmQryRespAccessDataVersion  1

/*-------------------------------------------------------------------------+
 |  Type definition for envSetUp parameter on dsmSetUp()
 +-------------------------------------------------------------------------*/
typedef struct tsmEnvSetUp
{
   dsUint16_t      stVersion;                     /* structure version */
   dsChar_t        dsmiDir[DSM_PATH_MAX + DSM_NAME_MAX +1];
   dsChar_t        dsmiConfig[DSM_PATH_MAX + DSM_NAME_MAX +1];
   dsChar_t        dsmiLog[DSM_PATH_MAX + DSM_NAME_MAX +1];
   char            **argv;  /* for executables name argv[0] */
   dsChar_t        logName[DSM_NAME_MAX +1];
   dsmBool_t       reserved1;         /* for future use */
   dsmBool_t       reserved2;                 /* for future use */
} tsmEnvSetUp;

#define tsmEnvSetUpVersion  4

/*-------------------------------------------------------------------------+
 |  Type definition for dsmInitExIn_t
 +-------------------------------------------------------------------------*/
typedef struct tsmInitExIn_t
{
   dsUint16_t         stVersion;                    /* structure version */
   tsmApiVersionEx    *apiVersionExP;
   dsChar_t           *clientNodeNameP;
   dsChar_t           *clientOwnerNameP;
   dsChar_t           *clientPasswordP;
   dsChar_t           *userNameP;
```

```
   dsChar_t           *userPasswordP;
   dsChar_t           *applicationTypeP;
   dsChar_t           *configfile;
   dsChar_t           *options;
   dsChar_t           dirDelimiter;
   dsmBool_t          useUnicode;
   dsmBool_t          bCrossPlatform;
   dsmBool_t          bService;
   dsmBool_t          bEncryptKeyEnabled;
   dsChar_t           *encryptionPasswordP;
   dsmBool_t          useTsmBuffers;
   dsUint8_t          numTsmBuffers;
   tsmAppVersion      appVersionP;
} tsmInitExIn_t;

#define tsmInitExInVersion 5

/*-------------------------------------------------------------------------+
 |  Type definition for dsmInitExOut_t
 +-------------------------------------------------------------------------*/
typedef struct tsmInitExOut_t
{
   dsUint16_t         stVersion;                    /* structure version */
   dsInt16_t          userNameAuthorities;
   dsInt16_t          infoRC;        /* error return code if encountered */
   /* adsm server name                */
   dsChar_t           adsmServerName[DSM_MAX_SERVERNAME_LENGTH+1];
   dsUint16_t         serverVer;     /* Server's version number          */
   dsUint16_t         serverRel;     /* Server's release number          */
   dsUint16_t         serverLev;     /* Server's level number            */
   dsUint16_t         serverSubLev;  /* Server's sublevel number         */
   dsmBool_t          bIsFailOverMode; /* true if failover has occured */
   dsChar_t           replServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* repl server name */
   dsChar_t           homeServerName[DSM_MAX_SERVERNAME_LENGTH+1]; /* home server name */
} tsmInitExOut_t;

#define tsmInitExOutVersion 3

/*-------------------------------------------------------------------------+
 |  Type definition for dsmLogExIn_t
 +-------------------------------------------------------------------------*/
typedef struct tsmLogExIn_t
{
   dsUint16_t         stVersion;  /* structure version */
   dsmLogSeverity     severity;
   dsChar_t           appMsgID[8];
   dsmLogType         logType;    /* log type : local, server, both */
   dsChar_t           *message;   /* text of message to be logged */
   dsChar_t           appName[DSM_MAX_PLATFORM_LENGTH];
   dsChar_t           osPlatform[DSM_MAX_PLATFORM_LENGTH];
   dsChar_t           appVersion[DSM_MAX_PLATFORM_LENGTH];

} tsmLogExIn_t;

#define tsmLogExInVersion 2

/*-------------------------------------------------------------------------+
 |  Type definition for dsmlogExOut_t
 +-------------------------------------------------------------------------*/
typedef struct tsmLogExOut_t
{
   dsUint16_t         stVersion;  /* structure version */
} tsmLogExOut_t;

#define tsmLogExOutVersion 1

/*-------------------------------------------------------------------------+
```

```
 |  Type definition for dsmRenameIn_t
 +--------------------------------------------------------------------------*/
typedef struct tsmRenameIn_t
{
   dsUint16_t       stVersion;                    /* structure version  */
   dsUint32_t       tsmHandle;                    /* handle for session */
   dsUint8_t        repository;                   /* Backup or Archive  */
   tsmObjName       *objNameP ;                   /* object name */
   dsChar_t         newHl[DSM_MAX_HL_LENGTH + 1];  /* new High level name */
   dsChar_t         newLl[DSM_MAX_LL_LENGTH + 1];  /* new Low level name */
   dsmBool_t        merge;                       /* merge into existing name*/
   ObjID            objId;                        /* objId for Archive */
} tsmRenameIn_t;

#define tsmRenameInVersion 1

/*--------------------------------------------------------------------------+
 |  Type definition for dsmRenameOut_t
 +--------------------------------------------------------------------------*/
typedef struct tsmRenameOut_t
{
   dsUint16_t         stVersion;                     /* structure version */
} tsmRenameOut_t;

#define tsmRenameOutVersion 1

/*--------------------------------------------------------------------------+
 |  Type definition for tsmEndSendObjExIn_t
 +--------------------------------------------------------------------------*/
typedef struct tsmEndSendObjExIn_t
{
   dsUint16_t       stVersion;                    /* structure version  */
   dsUint32_t       tsmHandle;                    /* handle for session */
} tsmEndSendObjExIn_t;

#define tsmEndSendObjExInVersion 1

/*--------------------------------------------------------------------------+
 |  Type definition for dsmEndSendObjExOut_t
 +--------------------------------------------------------------------------*/
typedef struct tsmEndSendObjExOut_t
{
   dsUint16_t         stVersion;           /* structure version */
   dsStruct64_t       totalBytesSent;      /* total bytes read from app */
   dsmBool_t          objCompressed;       /* was object compressed */
   dsStruct64_t       totalCompressSize;   /* total size after compress */
   dsStruct64_t       totalLFBytesSent;    /* total bytes sent Lan Free */
   dsUint8_t          encryptionType;      /* type of encryption used   */
   dsmBool_t          objDeduplicated;     /* was object processed for dist. data dedup */
   dsStruct64_t       totalDedupSize;      /* total size after de-dup */
}tsmEndSendObjExOut_t;

#define tsmEndSendObjExOutVersion 3

/*--------------------------------------------------------------------------+
 |  Type definition for tsmGroupHandlerIn_t
 +--------------------------------------------------------------------------*/
typedef struct tsmGroupHandlerIn_t
{
   dsUint16_t       stVersion;        /* structure version                     */
   dsUint32_t       tsmHandle;        /* handle for session                    */
   dsUint8_t        groupType;        /* Type of group                         */
   dsUint8_t        actionType;       /* Type of group operation               */
   dsUint8_t        memberType;       /* Type of member: Leader or member      */
   dsStruct64_t     leaderObjId;      /* OBJID of the groupleader              */
   dsChar_t         *uniqueGroupTagP; /* Unique group identifier               */
   tsmObjName       *objNameP ;       /* group leader object name              */
```

```
   dsmGetList       memberObjList;    /* list of objects to remove, assign       */
} tsmGroupHandlerIn_t;

#define tsmGroupHandlerInVersion 1

/*-------------------------------------------------------------------------+
 |  Type definition for tsmGroupHandlerExOut_t
 +-------------------------------------------------------------------------*/
typedef struct tsmGroupHandlerOut_t
{
   dsUint16_t         stVersion;                    /* structure version */
} tsmGroupHandlerOut_t;

#define tsmGroupHandlerOutVersion 1

/*-------------------------------------------------------------------------+
 |  Type definition for tsmEndTxnExIn_t
 +-------------------------------------------------------------------------*/
typedef struct tsmEndTxnExIn_t
{
   dsUint16_t       stVersion;                   /* structure version  */
   dsUint32_t       tsmHandle;                   /* handle for session */
   dsUint8_t        vote;
} tsmEndTxnExIn_t;

#define tsmEndTxnExInVersion 1

/*-------------------------------------------------------------------------+
 |  Type definition for tsmEndTxnExOut_t
 +-------------------------------------------------------------------------*/
typedef struct tsmEndTxnExOut_t
{
   dsUint16_t         stVersion;              /* structure version             */
   dsUint16_t         reason;                 /* reason code                   */
   dsStruct64_t       groupLeaderObjId;       /* groupLeader obj id returned on */
   /* DSM_ACTION_OPEN               */
    dsUint8_t          reserved1;              /* future use                    */
    dsUint16_t         reserved2;              /* future use                    */
} tsmEndTxnExOut_t;

#define tsmEndTxnExOutVersion 1

/*-------------------------------------------------------------------------+
 |  Type definition for tsmEndGetDataExIn_t
 +-------------------------------------------------------------------------*/
typedef struct tsmEndGetDataExIn_t
{
   dsUint16_t     stVersion;   /* structure version  */
   dsUint32_t     tsmHandle;   /* handle for session */
}tsmEndGetDataExIn_t;

#define tsmEndGetDataExInVersion 1

/*-------------------------------------------------------------------------+
 |  Type definition for tsmEndGetDataExOut_t
 +-------------------------------------------------------------------------*/
typedef struct tsmEndGetDataExOut_t
{
   dsUint16_t     stVersion;         /* structure version            */
   dsUint16_t     reason;            /* reason code                  */
   dsStruct64_t   totalLFBytesRecv;  /* total lan free bytes recieved */
}tsmEndGetDataExOut_t;

#define tsmEndGetDataExOutVersion 1

/*-------------------------------------------------------------------------+
 |  Type definition for  on tsmRetentionEvent()                            |
```

```
 +---------------------------------------------------------------------------*/
typedef struct tsmRetentionEventIn_t
{
   dsUint16_t         stVersion;                    /* structure version  */
   dsUint32_t         tsmHandle;                    /* session Handle     */
   dsmEventType_t     eventType;                    /* Event type         */
   dsmObjList_t       objList;                      /* object ID          */
}tsmRetentionEventIn_t;

#define tsmRetentionEventInVersion 1

/*---------------------------------------------------------------------------+
 |  Type definition for  on tsmRetentionEvent()               |
 +---------------------------------------------------------------------------*/
typedef struct tsmRetentionEventOut_t
{
   dsUint16_t       stVersion ;                   /* structure version      */
}tsmRetentionEventOut_t;

#define tsmRetentionEventOutVersion 1

/*---------------------------------------------------------------------------+
 |  Type definition for tsmUpdateObjExIn_t
 +---------------------------------------------------------------------------*/
typedef struct tsmUpdateObjExIn_t
{
   dsUint16_t      stVersion;            /* structure version   */
   dsUint32_t      tsmHandle;            /* session Handle      */
   tsmSendType     sendType;             /* send type back/arch */
   dsChar_t        *descrP;              /* archive description */
   tsmObjName      *objNameP;            /* objName             */
   tsmObjAttr      *objAttrPtr;          /* attribute           */
   dsUint32_t      objUpdAct;            /* update action       */
   ObjID           archObjId;            /* objId for archive   */
}tsmUpdateObjExIn_t;

#define tsmUpdateObjExInVersion 1

/*---------------------------------------------------------------------------+
 |  Type definition for tsmUpdateObjExOut_t
 +---------------------------------------------------------------------------*/
typedef struct tsmUpdateObjExOut_t
{
   dsUint16_t         stVersion;        /* structure version */
}tsmUpdateObjExOut_t;

#define tsmUpdateObjExOutVersion 1

#if _OPSYS_TYPE == DS_WINNT
#pragma pack()
#endif


#ifdef _MAC
#pragma options align = reset
#endif
#endif /* _H_TSMAPITD */

/*****************************************************************************
* Tivoli Storage Manager                                                     *
* API Client Component                                                       *
*                                                                            *
* (C) Copyright IBM Corporation 1993,2010                                    *
*****************************************************************************/

/*****************************************************************************
* Header File Name: dsmapips.h
```

```
*
* Environment:      ******************************************
*                   ** This is a platform-specific source file **
*                   ** versioned for Windows NT               **
*
*                   ******************************************
*
* Design Notes:     This file includes platform dependent definitions
*
* Descriptive-name: Definitions for Tivoli Storage Manager typedefs and LINKAGE
*-----------------------------------------------------------------------*/

#ifndef _H_DSMAPIPS
#define _H_DSMAPIPS


#ifndef _WIN64
#pragma pack(1)
#endif

/*<><><>
```

```
   typedef __int64              dsInt64_t;
   typedef unsigned __int64     dsUint64_t;
   /*=== A "true" unsigned 64-bit integer ===*/
   typedef __int64              dsLongLong_t;
#else
typedef struct tagUINT64_t
   {
      dsUint32_t hi;        /* Most significant 32 bits. */
      dsUint32_t lo;        /* Least significant 32 bits. */
   } dsUint64_t;
#endif

/*-------------------------------------------------------------------------+
| Type definition for bool_t                                               |
+-------------------------------------------------------------------------*/
/*
 *  Had to create a Boolean type that didn't clash with any other predefined
 *  version in any operating system or windowing system.
 */
typedef enum
{
   dsmFalse = 0x00,
   dsmTrue  = 0x01
}dsmBool_t ;


/*===  for backward compatability  ===*/
#define uint8    dsUint8_t
#define int8     dsInt8_t
#define uint16   dsUint16_t
#define int16    dsInt16_t
#define uint32   dsUint32_t
#define int32    dsInt32_t
#define uint64   dsStruct64_t
#define bool_t   dsBool_t
#define dsBool_t dsmBool_t
#define bTrue    dsmTrue
#define bFalse   dsmFalse

typedef struct
{
   dsUint32_t hi;        /* Most significant 32 bits. */
   dsUint32_t lo;        /* Least significant 32 bits. */
}dsStruct64_t ;

#endif /* DSMAPILIB */



#ifndef _WIN64
#pragma pack()
#endif
#endif  /* _H_DSMAPIPS */
/*************************************************************************
* Tivoli Storage Manager                                                 *
* Common Source Component                                                *
*                                                                        *
* (C) Copyright IBM Corporation 1993,2016                                *
*************************************************************************/

/*************************************************************************
* Header File Name: release.h
*
* Environment:     ***********************************************
*                  ** This is a platform-independent source file **
*                  ***********************************************
*
```

```
* Design Notes:    This file contains the common information about
*                  the actual version.release.level.sublevel
*
* Descriptive-name: Definitions for Tivoli Storage manager version
*
* Note: This file should contain no LOG or CMVC information. It is
*       shipped with the API code.
*
*-----------------------------------------------------------------------*/

#ifndef _H_RELEASE
#define _H_RELEASE

#define COMMON_VERSION         7
#define COMMON_RELEASE         1
#define COMMON_LEVEL           6
#define COMMON_SUBLEVEL        0
#define COMMON_DRIVER          dsTEXT("")

#define COMMON_VERSIONTXT "7.1.6.0"

#define SHIPYEARTXT  "2016"
#define SHIPYEARTXTW dsTEXT("2016")
#define TSMPRODTXT  "IBM Tivoli Storage Manager"

/*=====================================================================
   The following string definitions are used for VERSION information
   and should not be converted to dsTEXT or osTEXT.  They are used
   only at link time.

   These are also used when the Jar file is built on Unix.  See the
   the perl script tools/unx/mzbuild/createReleaseJava
=====================================================================*/
#define COMMON_VERSION_STR    "7"
#define COMMON_RELEASE_STR    "1"
#define COMMON_LEVEL_STR      "6"
#define COMMON_SUBLEVEL_STR   "0"
#define COMMON_DRIVER_STR     ""

/*=== product names definitions ===*/
#define COMMON_NAME_DFDSM      1
#define COMMON_NAME_ADSM       2
#define COMMON_NAME_TSM        3
#define COMMON_NAME_ITSM       4
#define COMMON_NAME            COMMON_NAME_ITSM

/*=====================================================================
   Internal version, release, and level (build) version.  This
   should be unique for every version+release+ptf of a product.
   This information is recorded in the file attributes and data
   stream for diagnostic purposes.
   NOTE: DO NOT MODIFY THESE VALUES. YOU CAN ONLY ADD NEW ENTRIES!
=====================================================================*/
#define COMMON_BUILD_TSM_510   1
#define COMMON_BUILD_TSM_511   2
#define COMMON_BUILD_TSM_515   3
#define COMMON_BUILD_TSM_516   4
#define COMMON_BUILD_TSM_520   5
#define COMMON_BUILD_TSM_522   6
#define COMMON_BUILD_TSM_517   7
#define COMMON_BUILD_TSM_523   8
#define COMMON_BUILD_TSM_530   9
#define COMMON_BUILD_TSM_524   10
#define COMMON_BUILD_TSM_532   11
#define COMMON_BUILD_TSM_533   12
#define COMMON_BUILD_TSM_525   13
#define COMMON_BUILD_TSM_534   14
```

```
#define COMMON_BUILD_TSM_540   15
#define COMMON_BUILD_TSM_535   16
#define COMMON_BUILD_TSM_541   17
#define COMMON_BUILD_TSM_550   18
#define COMMON_BUILD_TSM_542   19
#define COMMON_BUILD_TSM_551   20
#define COMMON_BUILD_TSM_610   21
#define COMMON_BUILD_TSM_552   22
#define COMMON_BUILD_TSM_611   23
#define COMMON_BUILD_TSM_543   24
#define COMMON_BUILD_TSM_620   25
#define COMMON_BUILD_TSM_612   26
#define COMMON_BUILD_TSM_553   27
#define COMMON_BUILD_TSM_613   28
#define COMMON_BUILD_TSM_621   29
#define COMMON_BUILD_TSM_622   30
#define COMMON_BUILD_TSM_614   31
#define COMMON_BUILD_TSM_623   32
#define COMMON_BUILD_TSM_630   33
#define COMMON_BUILD_TSM_615   34
#define COMMON_BUILD_TSM_624   35
#define COMMON_BUILD_TSM_631   36
#define COMMON_BUILD_TSM_640   37
#define COMMON_BUILD_TSM_710   38
#define COMMON_BUILD_TSM_625   39
#define COMMON_BUILD_TSM_641   40
#define COMMON_BUILD_TSM_711   41
#define COMMON_BUILD_TSM_712   42
#define COMMON_BUILD_TSM_713   43
#define COMMON_BUILD_TSM_714   44
#define COMMON_BUILD_TSM_642   47
#define COMMON_BUILD_TSM_643   48
#define COMMON_BUILD_TSM_715   49
#define COMMON_BUILD_TSM_716   50
#define COMMON_BUILD           COMMON_BUILD_TSM_716

/*=== define VRL as an Int for bitmap version compares ====*/
static const  int  VRL_712 = 712;
static const  int  VRL_713 = 713;
static const  int  VRL_714 = 714;
static const  int  VRL_715 = 715;
static const  int  VRL_716 = 716;

#define   TDP4VE_PLATFORM_STRING_MBCS     "TDP VMware"
#define   TDP4VE_PLATFORM_STRING          dsTEXT("TDP VMware")

#define   TDP4HYPERV_PLATFORM_STRING_MBCS "TDP HyperV"
#define   TDP4HYPERV_PLATFORM_STRING      dsTEXT("TDP HyperV")

#endif /* _H_RELEASE */
```

Appendix C. API function definitions source file

This appendix contains the dsmapifp.h header file, so you can see the function definitions for the API.

Note: **DSMLINKAGE** is defined differently for each operating system. See the definitions in the dsmapips.h file for your specific operating system.

The information that is provided here contains a point-in-time copy of the files that are distributed with the API. View the files in the API distribution package for the latest version.

```
/*****************************************************************
* Tivoli Storage Manager                                         *
* API Client Component                                           *
*                                                                *
* (C) Copyright IBM Corporation 1993,2002                        *
*****************************************************************/

/*****************************************************************/
/* Header File Name: dsmapifp.h                                  */
/*                                                               */
/* Descriptive-name: Tivoli Storage Manager API function prototypes */
/*****************************************************************/
#ifndef _H_DSMAPIFP
#define _H_DSMAPIFP

#if defined(__cplusplus)
extern "C" {
#endif

#ifdef DYNALOAD_DSMAPI

/* function will be dynamically loaded */
#include "dsmapidl.h"

#else

/* functions will be implicitly loaded from library */

/*===============================================================*/
/*              P U B L I C   F U N C T I O N S                  */
/*===============================================================*/


extern dsInt16_t DSMLINKAGE  dsmBeginGetData(
       dsUint32_t            dsmHandle,
       dsBool_t              mountWait,
       dsmGetType            getType,
       dsmGetList            *dsmGetObjListP
);

extern dsInt16_t DSMLINKAGE  dsmBeginQuery(
       dsUint32_t            dsmHandle,
       dsmQueryType          queryType,
       dsmQueryBuff          *queryBuffer
);

extern dsInt16_t DSMLINKAGE  dsmBeginTxn(
       dsUint32_t            dsmHandle
);
```

© Copyright IBM Corp. 1993, 2016

```
extern dsInt16_t DSMLINKAGE  dsmBindMC(
       dsUint32_t            dsmHandle,
       dsmObjName            *objNameP,
       dsmSendType           sendType,
       mcBindKey             *mcBindKeyP
);

extern dsInt16_t DSMLINKAGE  dsmChangePW(
       dsUint32_t            dsmHandle,
       char                  *oldPW,
       char                  *newPW
);

extern dsInt16_t DSMLINKAGE dsmCleanUp(
       dsBool_t              mtFlag
);


extern dsInt16_t DSMLINKAGE  dsmDeleteAccess(
       dsUint32_t            dsmHandle,
       dsUint32_t            ruleNum
);

extern dsInt16_t DSMLINKAGE  dsmDeleteObj(
       dsUint32_t            dsmHandle,
       dsmDelType            delType,
       dsmDelInfo            delInfo
);

extern dsInt16_t DSMLINKAGE  dsmDeleteFS(
       dsUint32_t            dsmHandle,
       char                  *fsName,
       dsUint8_t             repository
);

extern dsInt16_t DSMLINKAGE  dsmEndGetData(
       dsUint32_t            dsmHandle
);

extern dsInt16_t DSMLINKAGE  dsmEndGetDataEx(
       dsmEndGetDataExIn_t  *dsmEndGetDataExInP,
       dsmEndGetDataExOut_t *dsmEndGetDataExOutP
);

extern dsInt16_t DSMLINKAGE  dsmEndGetObj(
       dsUint32_t            dsmHandle
);

extern dsInt16_t DSMLINKAGE  dsmEndQuery(
       dsUint32_t            dsmHandle
);

extern dsInt16_t DSMLINKAGE  dsmEndSendObj(
       dsUint32_t            dsmHandle
);

extern dsInt16_t DSMLINKAGE  dsmEndSendObjEx(
       dsmEndSendObjExIn_t   *dsmEndSendObjExInP,
       dsmEndSendObjExOut_t  *dsmEndSendObjExOutP
);

extern dsInt16_t DSMLINKAGE  dsmEndTxnEx(
       dsmEndTxnExIn_t      *dsmEndTxnExInP,
       dsmEndTxnExOut_t     *dsmEndTxnExOutP
);

extern dsInt16_t DSMLINKAGE  dsmEndTxn(
```

```
       dsUint32_t            dsmHandle,
       dsUint8_t             vote,
       dsUint16_t            *reason
);

extern dsInt16_t DSMLINKAGE  dsmGetData(
       dsUint32_t            dsmHandle,
       DataBlk               *dataBlkPtr
);

extern dsInt16_t DSMLINKAGE  dsmGetBufferData(
       getBufferDataIn_t     *dsmGetBufferDataInP,
       getBufferDataOut_t    *dsmGetBufferDataOutP
);

extern dsInt16_t DSMLINKAGE  dsmGetNextQObj(
       dsUint32_t            dsmHandle,
       DataBlk               *dataBlkPtr
) ;

extern dsInt16_t DSMLINKAGE  dsmGetObj(
       dsUint32_t            dsmHandle,
       ObjID                 *objIdP,
       DataBlk               *dataBlkPtr
);

extern dsInt16_t DSMLINKAGE  dsmGroupHandler(
       dsmGroupHandlerIn_t   *dsmGroupHandlerInP,
       dsmGroupHandlerOut_t  *dsmGroupHandlerOutP
);

extern dsInt16_t DSMLINKAGE  dsmInit(
       dsUint32_t            *dsmHandle,
       dsmApiVersion         *dsmApiVersionP,
       char                  *clientNodeNameP,
       char                  *clientOwnerNameP,
       char                  *clientPasswordP,
       char                  *applicationType,
       char                  *configfile,
       char                  *options
);

extern dsInt16_t DSMLINKAGE  dsmInitEx(
       dsUint32_t            *dsmHandleP,
       dsmInitExIn_t         *dsmInitExInP,
       dsmInitExOut_t        *dsmInitExOutP
);


extern dsInt16_t DSMLINKAGE  dsmLogEvent(
       dsUint32_t            dsmHandle,
       logInfo               *lopInfoP
);

extern dsInt16_t DSMLINKAGE  dsmLogEventEx(
       dsUint32_t            dsmHandle,
       dsmLogExIn_t          *dsmLogExInP,
       dsmLogExOut_t         *dsmLogExOutP
);

extern dsInt16_t DSMLINKAGE  dsmQueryAccess(
       dsUint32_t            dsmHandle,
       qryRespAccessData     **accessListP,
       dsUint16_t            *numberOfRules
);

extern void DSMLINKAGE       dsmQueryApiVersion(
```

```
      dsmApiVersion         *apiVersionP
);

extern void DSMLINKAGE      dsmQueryApiVersionEx(
      dsmApiVersionEx       *apiVersionP
);

extern dsInt16_t DSMLINKAGE  dsmQueryCliOptions(
      optStruct             *optstructP
);

extern dsInt16_t DSMLINKAGE  dsmQuerySessInfo(
      dsUint32_t            dsmHandle,
      ApiSessInfo           *SessInfoP
);

extern dsInt16_t DSMLINKAGE  dsmQuerySessOptions(
      dsUint32_t            dsmHandle,
      optStruct             *optstructP
);

extern dsInt16_t DSMLINKAGE  dsmRCMsg(
      dsUint32_t            dsmHandle,
      dsInt16_t             dsmRC,
      char                  *msg
);

extern dsInt16_t DSMLINKAGE  dsmRegisterFS(
      dsUint32_t            dsmHandle,
      regFSData             *regFilespaceP
);

extern dsInt16_t DSMLINKAGE   dsmReleaseBuffer(
      releaseBufferIn_t     *dsmReleaseBufferInP,
      releaseBufferOut_t    *dsmReleaseBufferOutP
);

extern dsInt16_t DSMLINKAGE   dsmRenameObj(
      dsmRenameIn_t         *dsmRenameInP,
      dsmRenameOut_t        *dsmRenameOutP
);

extern dsInt16_t DSMLINKAGE  dsmRequestBuffer(
      requestBufferIn_t     *dsmRequestBufferInP,
      requestBufferOut_t    *dsmRequestBufferOutP
);

extern dsInt16_t DSMLINKAGE   dsmRetentionEvent(
      dsmRetentionEventIn_t  *dsmRetentionEventInP,
      dsmRetentionEventOut_t *dsmRetentionEventOutP
);

extern dsInt16_t DSMLINKAGE   dsmSendBufferData(
      sendBufferDataIn_t    *dsmSendBufferDataInP,
      sendBufferDataOut_t   *dsmSendBufferDataOutP
);

extern dsInt16_t DSMLINKAGE   dsmSendData(
      dsUint32_t            dsmHandle,
      DataBlk               *dataBlkPtr
) ;

extern dsInt16_t DSMLINKAGE   dsmSendObj(
      dsUint32_t            dsmHandle,
      dsmSendType           sendType,
      void                  *sendBuff,
      dsmObjName            *objNameP,
```

```
       ObjAttr               *objAttrPtr,
       DataBlk               *dataBlkPtr
);

extern dsInt16_t DSMLINKAGE   dsmSetAccess(
       dsUint32_t            dsmHandle,
       dsmAccessType         accessType,
       dsmObjName            *objNameP,
       char                  *node,
       char                  *owner
);

extern dsInt16_t DSMLINKAGE   dsmSetUp(
       dsBool_t              mtFlag,
       envSetUp              *envSetUpP
);

extern dsInt16_t DSMLINKAGE   dsmTerminate(
       dsUint32_t            dsmHandle
);

extern dsInt16_t DSMLINKAGE   dsmUpdateFS(
       dsUint32_t            dsmHandle,
       char                  *fs,
       dsmFSUpd              *fsUpdP,
       dsUint32_t            fsUpdAct
);

extern dsInt16_t DSMLINKAGE   dsmUpdateObj(
       dsUint32_t            dsmHandle,
       dsmSendType           sendType,
       void                  *sendBuff,
       dsmObjName            *objNameP,
       ObjAttr               *objAttrPtr,
       dsUint32_t            objUpdAct

);

extern dsInt16_t DSMLINKAGE  dsmUpdateObjEx(
       dsmUpdateObjExIn_t    *dsmUpdateObjExInP,
       dsmUpdateObjExOut_t   *dsmUpdateObjExOutP
);


#endif /* ifdef DYNALOAD */

#if defined(__cplusplus)
   }
#endif

#endif /* _H_DSMAPIFP */
```

This section contains the function definitions for the API. It is a copy of the `tsmapifp.h` header file.

Note: **`DSMLINKAGE`** is defined differently for each operating system. See the definitions in the `tsmapips.h` file for your specific operating system.

```
/*************************************************************************
* Tivoli Storage Manager                                 *
* API Client Component                                   *
*                                                        *
* (C) Copyright IBM Corporation 1993,2002                *
*************************************************************************/

/*************************************************************************/
/* Header File Name: tsmapifp.h                             */
```

```
/*                                                                  */
/* Descriptive-name: Tivoli Storage Manager API function prototypes       */
/********************************************************************/
#ifndef _H_TSMAPIFP
#define _H_TSMAPIFP

#if defined(__cplusplus)
extern "C" {
#endif

#ifdef DYNALOAD_DSMAPI

/* function will be dynamically loaded */
#include "dsmapidl.h"

#else

/* functions will be implicitly loaded from library */

/*==================================================================*/
/*P U B L I C   F U N C T I O N S                                   */
/*==================================================================*/


typedef void tsmQueryBuff;

extern dsInt16_t DSMLINKAGE  tsmBeginGetData(
       dsUint32_t            tsmHandle,
       dsBool_t              mountWait,
       tsmGetType            getType,
       dsmGetList            *dsmGetObjListP
);

extern dsInt16_t DSMLINKAGE  tsmBeginQuery(
       dsUint32_t            tsmHandle,
       tsmQueryType          queryType,
       tsmQueryBuff          *queryBuffer
);

extern dsInt16_t DSMLINKAGE  tsmBeginTxn(
       dsUint32_t            tsmHandle
);

extern dsInt16_t DSMLINKAGE  tsmBindMC(
       dsUint32_t            tsmHandle,
       tsmObjName            *objNameP,
       tsmSendType           sendType,
       tsmMcBindKey          *mcBindKeyP
);

extern dsInt16_t DSMLINKAGE  tsmChangePW(
       dsUint32_t            tsmHandle,
       dsChar_t              *oldPW,
       dsChar_t              *newPW
);

extern dsInt16_t DSMLINKAGE tsmCleanUp(
       dsBool_t              mtFlag
);


extern dsInt16_t DSMLINKAGE  tsmDeleteAccess(
       dsUint32_t            tsmHandle,
       dsUint32_t            ruleNum
);

extern dsInt16_t DSMLINKAGE  tsmDeleteObj(
```

```
       dsUint32_t            tsmHandle,
       tsmDelType            delType,
       tsmDelInfo            delInfo
);

extern dsInt16_t DSMLINKAGE  tsmDeleteFS(
       dsUint32_t            tsmHandle,
       dsChar_t              *fsName,
       dsUint8_t             repository
);

extern dsInt16_t DSMLINKAGE  tsmEndGetData(
       dsUint32_t            tsmHandle
);

extern dsInt16_t DSMLINKAGE  tsmEndGetDataEx(
       tsmEndGetDataExIn_t      *tsmEndGetDataExInP,
       tsmEndGetDataExOut_t     *tsmEndGetDataExOutP
);

extern dsInt16_t DSMLINKAGE  tsmEndGetObj(
       dsUint32_t            tsmHandle
);

extern dsInt16_t DSMLINKAGE  tsmEndQuery(
       dsUint32_t            tsmHandle
);

extern dsInt16_t DSMLINKAGE  tsmEndSendObj(
       dsUint32_t            tsmHandle
);

extern dsInt16_t DSMLINKAGE  tsmEndSendObjEx(
       tsmEndSendObjExIn_t      *tsmEndSendObjExInP,
       tsmEndSendObjExOut_t     *tsmEndSendObjExOutP
);

extern dsInt16_t DSMLINKAGE  tsmEndTxn(
       dsUint32_t            tsmHandle,
       dsUint8_t             vote,
       dsUint16_t            *reason
);

extern dsInt16_t DSMLINKAGE  tsmEndTxnEx(
       tsmEndTxnExIn_t      *tsmEndTxnExInP,
       tsmEndTxnExOut_t     *tsmEndTxnExOutP
);

extern dsInt16_t DSMLINKAGE  tsmGetData(
       dsUint32_t            tsmHandle,
       DataBlk*dataBlkPtr
);

extern dsInt16_t DSMLINKAGE  tsmGetBufferData(
       getBufferDataIn_t     *tsmGetBufferDataInP,
       getBufferDataOut_t    *tsmGetBufferDataOutP
);

extern dsInt16_t DSMLINKAGE  tsmGetNextQObj(
       dsUint32_t            tsmHandle,
       DataBlk*dataBlkPtr
) ;

extern dsInt16_t DSMLINKAGE  tsmGetObj(
       dsUint32_t            tsmHandle,
       ObjID                 *objIdP,
       DataBlk               *dataBlkPtr
```

```
);

extern dsInt16_t DSMLINKAGE  tsmGroupHandler(
       tsmGroupHandlerIn_t   *tsmGroupHandlerInP,
       tsmGroupHandlerOut_t  *tsmGroupHandlerOutP
);

extern dsInt16_t DSMLINKAGE  tsmInitEx(
       dsUint32_t            *tsmHandleP,
       tsmInitExIn_t         *tsmInitExInP,
       tsmInitExOut_t        *tsmInitExOutP
);


extern dsInt16_t DSMLINKAGE  tsmLogEventEx(
       dsUint32_t            tsmHandle,
       tsmLogExIn_t          *tsmLogExInP,
       tsmLogExOut_t         *tsmLogExOutP
);

extern dsInt16_t DSMLINKAGE  tsmQueryAccess(
       dsUint32_t            tsmHandle,
       tsmQryRespAccessData  **accessListP,
       dsUint16_t            *numberOfRules
);


extern void DSMLINKAGE       tsmQueryApiVersionEx(
       tsmApiVersionEx       *apiVersionP
);

extern dsInt16_t DSMLINKAGE  tsmQueryCliOptions(
       tsmOptStruct          *optstructP
);

extern dsInt16_t DSMLINKAGE  tsmQuerySessInfo(
       dsUint32_t            tsmHandle,
       tsmApiSessInfo        *SessInfoP
);

extern dsInt16_t DSMLINKAGE  tsmQuerySessOptions(
       dsUint32_t            tsmHandle,
       tsmOptStruct          *optstructP
);

extern dsInt16_t DSMLINKAGE  tsmRCMsg(
       dsUint32_t            tsmHandle,
       dsInt16_t             tsmRC,
       dsChar_t              *msg
);

extern dsInt16_t DSMLINKAGE  tsmRegisterFS(
       dsUint32_t            tsmHandle,
       tsmRegFSData          *regFilespaceP
);

extern dsInt16_t DSMLINKAGE  tsmReleaseBuffer(
       releaseBufferIn_t     *tsmReleaseBufferInP,
       releaseBufferOut_t    *tsmReleaseBufferOutP
);

extern dsInt16_t DSMLINKAGE  tsmRenameObj(
       tsmRenameIn_t         *tsmRenameInP,
       tsmRenameOut_t        *tsmRenameOutP
);

extern dsInt16_t DSMLINKAGE  tsmRequestBuffer(
```

```
       requestBufferIn_t     *tsmRequestBufferInP,
       requestBufferOut_t    *tsmRequestBufferOutP
);

extern dsInt16_t DSMLINKAGE   tsmRetentionEvent(
       tsmRetentionEventIn_t  *tsmRetentionEventInP,
       tsmRetentionEventOut_t *tsmRetentionEventOutP
);

extern dsInt16_t DSMLINKAGE   tsmSendBufferData(
       sendBufferDataIn_t     *tsmSendBufferDataInP,
       sendBufferDataOut_t    *tsmSendBufferDataOutP
);

extern dsInt16_t DSMLINKAGE  tsmSendData(
       dsUint32_t             tsmHandle,
       DataBlk                *dataBlkPtr
);

extern dsInt16_t DSMLINKAGE  tsmSendObj(
       dsUint32_t             tsmHandle,
       tsmSendType            sendType,
       void                   *sendBuff,
       tsmObjName             *objNameP,
       tsmObjAttr             *objAttrPtr,
       DataBlk                *dataBlkPtr
);

extern dsInt16_t DSMLINKAGE  tsmSetAccess(
       dsUint32_t             tsmHandle,
       tsmAccessType          accessType,
       tsmObjName             *objNameP,
       dsChar_t               *node,
       dsChar_t               *owner
);

extern dsInt16_t DSMLINKAGE tsmSetUp(
       dsBool_t              mtFlag,
       tsmEnvSetUp           *envSetUpP
);

extern dsInt16_t DSMLINKAGE  tsmTerminate(
       dsUint32_t             tsmHandle
);

extern dsInt16_t DSMLINKAGE  tsmUpdateFS(
       dsUint32_t             tsmHandle,
       dsChar_t               *fs,
       tsmFSUpd               *fsUpdP,
       dsUint32_t             fsUpdAct
);

extern dsInt16_t DSMLINKAGE  tsmUpdateObj(
       dsUint32_t             tsmHandle,
       tsmSendType            sendType,
       void                   *sendBuff,
       tsmObjName             *objNameP,
       tsmObjAttr             *objAttrPtr,
       dsUint32_t             objUpdAct

);

extern dsInt16_t DSMLINKAGE tsmUpdateObjEx(
       tsmUpdateObjExIn_t         *tsmUpdateObjExInP,
       tsmUpdateObjExOut_t        *tsmUpdateObjExOutP
);
```

```
#endif /* ifdef DYNALOAD */

#if defined(__cplusplus)
   }
#endif

#endif /* _H_TSMAPIFP */
```

Appendix D. Accessibility features for the Tivoli Storage Manager product family

Accessibility features help users who have a disability, such as restricted mobility or limited vision to use information technology products successfully.

Accessibility features

The IBM Tivoli Storage Manager family of products includes the following accessibility features:

- Keyboard-only operation using standard operating-system conventions
- Interfaces that support assistive technology such as screen readers

The command-line interfaces of all products in the product family are accessible.

Tivoli Storage Manager Operations Center provides the following additional accessibility features when you use it with a Mozilla Firefox browser on a Microsoft Windows system:

- Screen magnifiers and content zooming
- High contrast mode

The Operations Center and the Tivoli Storage Manager server can be installed in console mode, which is accessible.

The Operations Center help system is enabled for accessibility. For more information, click the question mark icon on the help system menu bar.

Vendor software

The Tivoli Storage Manager product family includes certain vendor software that is not covered under the IBM license agreement. IBM makes no representation about the accessibility features of these products. Contact the vendor for the accessibility information about its products.

IBM and accessibility

See the IBM Human Ability and Accessibility Center (http://www.ibm.com/able) for information about the commitment that IBM has to accessibility.

© Copyright IBM Corp. 1993, 2016

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

© Copyright IBM Corp. 1993, 2016

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

```
Portions of this code are derived from IBM® Corp. Sample Programs.
© Copyright IBM® Corp. _enter the year or years_. All rights reserved.
```

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Linear Tape-Open, LTO, and Ultrium are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability
These terms and conditions are in addition to any terms of use for the IBM website.

Personal use
You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use
You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights Except as expressly granted in this permission, no other permissions,

licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies,” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Glossary

A glossary is available with terms and definitions for the IBM Tivoli Storage Manager family of products.

See Tivoli Storage Manager glossary.

To view glossaries for other IBM products, see IBM Terminology.

© Copyright IBM Corp. 1993, 2016

Index

Numerics

128–bit AES encryption support 46
256-bit AES encryption support 46
64-bit
  compiling 1
  requirements 1

A

access to objects
  by user 25
accessibility features 207
accessing to objects
  across nodes 25
active copies of objects 41
active version
  deleting 73
administrative user
  creating 22
administrator options 2
API
  dsmInitEx
    configuration file used by 2
  environment setup 3
  option string used by dsmInitEx 2
  overview 1
  sample applications 5
  using Unicode 83
API configuration file
  used by dsmInitEx 17
API options list
  used by dsmInitEx 17
application type 17, 113, 116
application version vii
archive copy group 28
archive files
  how long retained 28
archive objects
  expiration 30
  release 30
  suspend 30
archiveretentionprotection 31
archiving objects 41
asnodename 80
authorization rule
  dsmDeleteAccess function 97
authorized user 21, 25
automated client failover 55

B

backing up objects 41
backup
  multiple nodes 80
  using client node proxy 80
backup copy group 28
backup-archive client
  interoperability 77
buffer copy elimination
  overview 44
buffer copy elimination *(continued)*
  restore and retrieve 45

C

callbuff
  TSM buffer sample API
    applications 5
callevnt
  event-based retention 5
callhold
  detention hold sample API
    applications 5
callmt*
  multithreaded sample API
    applications 5
callmt1.c
  sample 16
callret
  data retention protection sample API
    applications 5
capacity
  file space 26
character sets 83
client node proxy support 80
client owner authority 22
client performance monitor 38
client performance monitor options
  PERFCOMMTIMEOUT 40
  PERFMONTCPPORT 40
  PERFMONTCPSERVERADDRESS 39
client-side data deduplication 52
code pages 83
commands
  makemtu 83
compatibility
  between versions of API 14
compiling
  Unicode 83
compressalways 2
  option 6
compression 42, 66
compression type
  LZ4 43
  LZW 43
configuration file
  API 2
configuration sources
  priority sequence 2
copy group 28
CTRL+C 16

D

dapi*
  single-threaded, interactive sample
    API applications 5
data deduplication 50
data deduplication files
  exclude 54
data deduplication files *(continued)*
  include 55
data protection 32
data retention 32
data structures
  size limits 14, 35
  version control 15
data transfer
  LAN-free 37
DB Chg operation 11
DBCS 83
delete archive 79
delete filespace 79
design recommendations 11
dir
  object type 24
disability 207
double-byte character set 83
dscenu.txt 3
dsierror.log 3
dsierror.log. 3
DSM_MAX_PLATFORM_LENGTH 17
dsm.opt 1
  asnodename option 80
  enablearchiveretentionprotection 32
  encryptkey 47
dsm.sys 1, 3, 19
  asnodename option 80
  enablearchiveretentionprotection 32
  encryptkey 47
dsmapifp.h
  header file 85, 197
dsmapips.h header file 155
dsmapitd.h 14, 112, 115
  header file 122
dsmapitd.h header file 155
dsmApiVersion function
  session 17
dsmBeginGetData function 66, 70
  buffer management 45
  code example 72
  dsmEndGetData function 99
  dsmTerminate function 99
  in flowchart 71
  overview 87
  return codes 88
  state diagram 70, 74
  syntax 88
dsmBeginQuery function
  dsmEndQuery function 101
  dsmGetNextQObj function 107
  flowchart 33
  management class 29
  overview 89
  querying 33
  querying example 34
  receiving data 67
  return codes 93
  sending data example 36
  state diagram 33, 74
  syntax 89

© Copyright IBM Corp. 1993, 2016

dsmBeginTxn 25
dsmBeginTxn function
  buffer copy elimination 44
  code example 61
  deleting objects 73
  deletion 30
  dsmEndTxn function 102
  dsmRenameObj function 128
  dsmRetentionEvent function 131
  expiration 30
  overview 93
  retention policy 32
  return codes 94
  state diagram 74
  syntax 94
  transaction model 36
dsmBindMC
  example 30
dsmBindMC function
  buffer copy elimination 44
  code example 61
  dsmSendObj function 134
  general description 60
  include-exclude list 29
  information returned by 29
  management classes 30
  object names 24
  overview 94
  return codes 95
  state diagram 74
  syntax 94
dsmChangePW
  general description 74
dsmChangePW function
  overview 95
  return codes 96
  session security 18
  state diagram 74
  syntax 96
dsmCleanUp function
  dsmSetUp function 138
  multithreading 16
  overview 96
  signals 16
  syntax 96
dsmclientV3.cat 3
dsmDeleteAccess function
  accessing objects 25
  overview 97
  syntax 97
dsmDeleteFS function
  example code 26
  file spaces 26
  file system management 27
  overview 97
  return codes 98
  state diagram 74
  syntax 97
dsmDeleteObj function
  deleting objects 73
  dsmEndTxn function 102
  dsmSendObj function
    management class 11
  object naming 11
  objects 41
  overview 98
  return codes 99
dsmDeleteObj function *(continued)*
  state diagram 74
  syntax 98
dsmEndGetData
  stopping process 70
dsmEndGetData function 66
  buffer management 45
  code example 72
  in flowchart 71
  LAN-free 37
  overview 99
  state diagram 70, 74
  syntax 100
dsmEndGetDataEx function
  overview 100
  syntax 100
dsmEndGetObj function 66
  buffer management 45
  code example 72
  dsmBeginGetData function 87
  in flowchart 71
  overview 100
  return codes 101
  state diagram 70, 74
  syntax 100
dsmEndQuery 33
  general description 33
dsmEndQuery function 34
  dsmGetNextQObj function 107
  flowchart 33
  overview 101
  querying the server 67
  state diagram 33, 74
  syntax 101
dsmEndSendObj function
  code example 61
  dsmEndTxn function 102
  dsmSendData function 133
  dsmSendObj function 134
  flowchart 59
  overview 101
  return codes 101
  sending objects 41
  state diagram 58, 74
  syntax 101
dsmEndSendObjEx function 44
  compression 42
  encryption 47
  LAN-free 37
  overview 102
  return codes 102
  syntax 102
dsmEndTxn function 30, 131
  buffer copy elimination 44
  code example 61
  deleting objects 73
  dsmEndTxnEx function 104
  dsmRenameObj function 128
  dsmRetentionEvent function 131
  dsmSendObj function 134
  file grouping 63
  flowchart 59
  overview 102
  return codes 103
  simultaneous-write operations 37
  state diagram 58, 74
  syntax 103
dsmEndTxn function *(continued)*
  transaction model 36
dsmEndTxnEx function
  file grouping 63
  overview 104
  return codes 105
  syntax 104
dsmEventType function
  retention policy 32
dsmGetBufferData function 45
  overview 106
  return codes 106
  syntax 106
dsmGetData 70
dsmGetData function
  code example 72
  in flowchart 71
  in state diagram 70
  overview 105
  return codes 105
  state diagram 74
  syntax 105
dsmGetDataEx function
  dsmReleaseBuffer function 128
  dsmRequestBuffer function 130
dsmGetList function
  dsmGetObj function 110
dsmGetNextObj
  dsmDeleteObj function 98
dsmGetNextQObj 33
  dsmEndQuery function 101
dsmGetNextQObj function 31, 33, 56
  dsmRetentionEvent function 131
  flowchart 33
  overview 107
  querying example 34
  return codes 109
  state diagram 33, 74
  syntax 107
dsmGetObj
  receiving objects 70
dsmGetObj function 66
  code example 72
  dsmBeginGetData function 87
  dsmEndGetObj function 100
  dsmGetData function 105
  in flowchart 71
  overview 110
  return codes 111
  state diagram 70, 74
  syntax 110
dsmGroupHandler function
  dsmEndTxnEx function 104
  file grouping 63
  overview 111
  return codes 112
  syntax 111
dsmgrp.c 65
dsmgrp*
  logical object grouping sample API
    applications 5
dsmHandle 124, 125
dsmHandle function
  session 17
DSMI_CONFIG environment variable 3
DSMI_DIR
  environment variable 6

DSMI_DIR environment variable 3
DSMI_LOG environment variable 3
dsmInit function
  overview 112
  retention protection 31
  return codes 114
  syntax 112
dsmInitEx function 25, 44
  administrative user 22
  asnodename option 80
  dsmChangePW function 95
  dsmEndGetData function 99
  dsmGetBufferData function 106
  dsmGetNextQObj function 107
  dsmLogEvent function 119
  dsmQueryCliOptions function 123
  dsmQuerySessOptions 125
  dsmReleaseBuffer function 128
  dsmSetUp function 138
  encryption 47
  expired password 18
  interoperability 80
  multithreading 16
  option string 2
  overview 115
  retention protection 31
  return codes 118
  session 17
  session owner, set 25
  session security 18
  specifying options 2
  starting session 17
  state diagram 74
  syntax 115
dsmIntitEx function
  dsmQuerySessInfo function 124
dsmLogEvent function
  overview 119
  return codes 120
  syntax 119
dsmLogEventEx function 74
  overview 120
  return codes 121
  syntax 120
dsmQuery function
  multiple nodes 80
dsmQueryAccess function 25
  dsmDeleteAccess function 97
  overview 121
dsmQueryApiVersion function
  overview 122
  state diagram 74
  syntax 122
dsmQueryApiVersionEx function
  overview 122
  syntax 122
  version control 14
dsmQueryAPIVersionEx function
  multithreading 16
dsmQueryCliOptions function
  dsmQuerySessOptions 125
  overview 123
  session 17
  syntax 123
dsmQuerySessInfo
  dsmDeleteFS function 97
dsmQuerySessInfo function
  dsmRetentionEvent function 131
  general description 18
  overview 124
  return codes 124
  state diagram 74
  syntax 124
  transaction model 36
dsmQuerySessOptions function
  overview 125
  syntax 125
dsmrc.h
  header file 145
dsmRCMsg function
  overview 126
  return codes 126
  syntax 126
dsmRegisterFS function
  example code 26
  file spaces 26
  overview 127
  return codes 127
  state diagram 74
  syntax 127
dsmReleaseBuffer function 44, 45
  dsmGetBufferData function 106
  dsmReleaseBuffer function 128
  dsmRequestBuffer function 130
  dsmSendBufferData function 132
  overview 128
  return codes 128
  syntax 128
dsmRenameObj function
  overview 128
  return codes 129
  syntax 129
dsmRequestBuffer function
  buffer copy elimination 44
  overview 130
  return codes 130
  syntax 130
dsmRetentionEvent function
  deletion 30
  expiration 30
  overview 131
  retention policy 32
  return codes 132
  syntax 131
dsmSendBufferData function
  buffer copy elimination 44
  overview 132
  return codes 133
  syntax 132
dsmSendData function
  code example 61
  compression 42
  dsmEndSendObj function 101
  dsmEndTxn function 102
  dsmSendObj function 134
  flowchart 59
  multithreading 16
  overview 133
  performance 38
  return codes 133
  sending objects 41
  state diagram 58, 74
  syntax 133
dsmSendObj
  retention policy 32
dsmSendObj function 32
  accessing objects 25
  backup copy group 29
  code example 61
  compression 42
  copy groups 29
  dsmEndTxn function 102
  flowchart 59
  in state diagram 58
  object naming 11
  overview 134
  retention policy 32
  sending objects 41
  state diagram 74
  syntax 135
dsmSendObjfunction
  deleting objects 73
dsmSendType function
  updating objects 73
dsmSetAccess function
  accessing objects 25
  overview 137
  return codes 137
  syntax 137
dsmSetUp function
  LAN-free 11, 37
  multithread 16
  multithreading 16, 37
  overview 138
  passwordaccess 21
  syntax 139
dsmtca
  version control 14
dsmTerminate 70
dsmTerminate function
  buffer 44
  buffer copy elimination 44
  dsmInit function 112
  dsmReleaseBuffer function 128
  dsmRequestBuffer function 130
  dsmSetUp function 138
  general description 18
  overview 140
  session 17
  signals 16
  state diagram 74
  syntax 140
dsmUpdateFS function
  example code 26
  file space management 26
  file spaces 26
  overview 140
  return codes 141
  state diagram 74
  syntax 140
dsmUpdateObj function
  change management class 28
  overview 141
  return codes 142
  syntax 141
dsmUpdateObject(Ex) function
  updating objects 73
dsmUpdateObjEx function
  change management class 28
  overview 143

dsmUpdateObjEx function *(continued)*
return codes 144
syntax 143

E

enablearchiveretentionprotection 32
dsm.opt 32
dsm.sys 32
encryption
application managed 47
authentication setting 46
interoperability 79
transparent 49
encryption and compression using buffer copy elimination 46
encryptkey 47
ending a session 17
with dsmTerminate 18
environment
setting up API 3
environment variables
by operating system 3
DSMI_CONFIG 3
DSMI_DIR 3
DSMI_LOG 3
envSetUp 139
errorlogretention
when to use 74
event
eventRetentionActivate 32
event logging 74
event-based
retention policy 32
eventRetentionActivate event 32
exclude data deduplication files 54
exclude objects 24

F

failover
overview 55
status information 56
fast path 33
fast path queries 89
file aggregation 37
file grouping 63
file space
capacity 26
deleting 26
managing 26
registering 26
file space management
dsmUpdateFS 26
file space name
file aggregation 37
overview 23
file spaces
non-Unicode 83
file system management
dsmDeleteFS 27
files
configuration 1
object type 24
option 1
flowchart
backup and archive example 58
restore and retrieve 70
fromowner option 26
function calls
short descriptions 85
function definitions, API 197, 201

G

group leader 63

H

header file dsmapips.h 155
header file dsmapitd.h 155
header file release.h 155
header file tsmapitd.h 155
header files
dsmapifp.h 197
dsmrc.h 145
tsmapifp.h 201
high-level names
dsmRenameObj function 128
high-level qualifier 77
HP thread stack 16

I

IBM Knowledge Center v
inactive copies of objects 41
include data deduplication files 55
include objects 24
include-exclude
file 140
include-exclude list 29, 84
InSession state 119, 120
interoperability
access to API objects 77
backup-archive client 77
commands 79
conventions
UNIX or Linux 77
Windows 77
naming API objects 77
operating system 80

K

keyboard 207
Knowledge Center v

L

LAN-free
data transfer 37
dsmEndGetDataEX function 100
dsmSetUp function 11
logging events 74
low-level names
dsmRenameObj function 128
low-level qualifier 77
LZ4 compression 43
LZW compression 43

M

makemtu 83
management class
associating objects 28
binding and rebinding to files 29
dsmBindMC, assigned by 29
querying 30
mbcs 83
messages
dsmRCMsg function 126
metadata
object naming 23
multithreading
flag 11
mtflag value 16
multithread option 16
overview 16
restrictions 16

N

node replication 55
nodes
accessing across owners 25
authorization 73
names 11
querying management classes 30
with client proxy support 80
NULL
backup or archive group 28

O

object
version control 41
object ids, overview 23
object naming
dsmBindMC 24
examples by OS 24
file space name 23
high-level
object name 24
interoperability 77
low-level
object name 24
object type 24
overview 23
object types 24
objectID values 11
objects
access rules 25
active copies 41
deleting 73
deleting from server 73
expiration cycle 73
inactive copies 41
turning off 73
updating 73
operating system interoperability 80
option list
format 114, 117
option string
API 2
fromowner 26
options
compressalways 2

options *(continued)*
enablearchiveretentionprotection 32
errorlogretention 74
fromnode 25
fromowner 25
not supported on API 1
passwordaccess 16, 112
servername 2
set by administrator 2
tcpbuffsize 38
tcpnodelay 38
tcpserveraddr 2
options files
user 3
owner authority 22
owner name 11, 25
NULL 25

P

partial object restore or retrieve 66
passwordaccess
generate 140
option 7, 11, 47
passwordaccess option
dsmInit function 112
generate 18
multithreading 16
userNamePswd value 22
without TCA 21
passwordaccess prompt 18
passworddir option
in dsm.sys 21
path examples
by OS 24
path information
interoperability 77
PERFMONCOMMTIMEOUT 40
PERFMONTCPPORT 40
PERFMONTCPSERVERADDRESS 39
performance considerations 38
dsmSendData function 38
performance monitor
client 38
policies to store data 28
policy
retention policy 32
proxynode 81
publications v

Q

qMCData structure 34
qryRespArchiveData 31
qryRespBackupData
dsmDeleteObj function 98
qryRespBackupData structure 33
queries, system 33
query
actlog 119
command 79
nodes with client proxy node
authority 80

R

rcApiOut
example, details 18
rcApiOut function
session 17
receiving data from a server
general description 65
partial object restore or retrieve 66
procedure 66
recommendations
dsmGetObject
large amounts of data 110
setting HP thread stack 16
registering file spaces 26
release.h header file 155
replication status 56
restore 79
objects from a server 65
restrictions
encryption and compression using
buffer copy elimination 46
multithreading 16
retention protection 31
retrieve 79
objects from a server 65
return codes
obtaining through dsmRCMsg 126
source header file 145

S

sample API applications
callbuff 5
callbuff - TSM buffer 5
callevnt 5
callevnt - event-based retention 5
callhold 5
callhold - detention hold 5
callmt* 5
callmt* - multithreaded sample API
applications 5
callmtu1.c 83
callmtu2.c 83
callret 5
callret - data retention protection
sample API applications 5
dapi* 5
dapi* - interactive, single-threaded 5
dsmgrp 5
dsmgrp* - object grouping sample 5
UNIX or Linux 5
Windows 32-bit 7
Windows 64-bit 8
sample application
callmt1.c 16
sample code
dsmgrp.c 65
security 18
selecting objects
to restore 67
sending data
to non-Unicode file spaces 83
sending data to a server 36
server
deleting objects from 73
server-side data deduplication 55
servername 2
session
password
session 18
security 18
starting with dsmInitEx 17
set access 79
sign-on process 18
signal handlers 16
signals, using 16
simultaneous-write operations
storage pools 37
size estimates 41
size limits
API data structures 14, 35
sizing objects 41
sorting objects
by restore order 67
starting a session 17
state
InSession 120
state diagram
backup and archive example 58
restore and retrieve 70
stopping a session 17
storage pools
simultaneous-write operations 37
structure
qryRespBackupData 33
qryRespFSData function 26
structures
qMCData 34
size limits 14, 35
system queries 33

T

target nodes and traditional nodes 80
TCA
session security 18
signals 17
version control 14
without passwordaccess 21
TCPport 19
TCPserver address 19
tcpserveraddr 2
transaction model
dsmBeginTxn function 93
Trusted Communication Agent
passwordaccess 21
session security 18
signals 17
tsmapifp.h 83
tsmapifp.h header file 201
tsmapitd.h 83
tsmapitd.h header file 155
turning off objects 73

U

Unicode
mbcs 83
non-Unicode file spaces 83
setting up 83
Windows 83

UNIX or Linux
sample API application 5
user
intervention 16

V

version control
API data structures 15
dsmQueryApiVersionEx, using 14
managing backed-up copies 41
versions
files retained 28

W

Windows 32-bit
sample application 7
Windows 64-bit
sample application 8

Product Number: 5608-E01
5608-E02
5608-E03

Printed in USA